hyväksymispäivä arvosana

arvostelija

Ammattikäyttöön tarkoitetun verkkosovelluksen käyttöliittymän käytettävyys, suunnittelu ja toteutus

Minna Sarakontu

Helsinki 29.4.2019

Pro gradu -tutkielma

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Minna Sarakontu

Työn nimi — Arbetets titel — Title

Ammattikäyttöön tarkoitetun verkkosovelluksenkäyttöliittymän käytettävyys, suunnittelu ja toteutus

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Pro gradu -tutkielma	29.4.2019	78 sivua + 7 liitesivua

Tiivistelmä — Referat — Abstract

Verkkosovelluksilla ja muilla sähköisillä järjestelmillä voidaan säästää resursseja, kun esimerkiksi työaikakirjanpito siirretään paperilta mobiiliin ja tietokoneen näytölle. Työntekoa tehostava sovellus vaatii kuitenkin hyvän käytettävyyden. Käytettävyydellä tarkoitetaan muun muassa sovelluksen käytön opittavuutta, tehokkuutta ja miellyttävyyttä.

Tämän tutkielman lähtökohtana on toimeksianto yritykseltä, jolla on tarve uudistaa työaikasovelluksen käyttöliittymää. Nykyisen käyttöliittymän suurin puute on sen skaalautumattomuus työntekijöiden määrän kasvaessa. Lisäksi käyttöliittymä on toteutettu vanhanaikaisilla tekniikoilla, eikä sen visuaalinen ulkoasu vastaa yrityksen muiden sovellusten tyylisuuntaa.

Tutkielmassa suunniteltiin ja toteutettiin työaikasovellukselle uuden käyttöliittymän prototyyppi, joka tarjoaa ratkaisut nykyisen käyttöliittymän ongelmakohtiin. Prototyypin suunnittelussa ja arvioinnissa hyödynnettiin kirjallisuuskatsauksessa kerättyä tietoa ihmisen ja koneen välisestä vuorovaikutuksesta, käytettävyydestä ja verkkosovellusten suunnittelusta.

Prototyypissä onnistuttiin toteuttamaan tuhansille työntekijätiedoille skaalautuva listanäkymä sekä kaikki nykyisen käyttöliittymän ominaisuudet modernimmalla ulkoasulla ja kehitystyökaluilla. Vasteaika- ja käytettävyystestauksessa havaittiin jonkin verran puutteita, jotka huomioidaan uuden käyttöliittymän jatkokehityksessä.

ACM Computing Classification System (CCS):
A.1 [Introductory and Survey],
I.7.m [Document and text processing]

Avainsanat — Nyckelord — Keywords
verkkosovelluskehitys, käyttöliittymäsuunnittelu, käytettävyys, kognitiotiede, JavaScript

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Esipuhe

Verkkosovelluksista ja niiden uusimmista teknologioista puhuminen ei ole järin hyvä aihe sellaiselle pro gradu -työlle, jota aikoo työstää vuoden päivät ja vähän ylikin. JavaScript-kirjastoja ja -sovelluskehyksiä luodaan, uudistetaan ja merkataan vanhentuneiksi sellaista vauhtia, etteivät alkuvuonna 2018 keräämäni tutkimustulokset niistä ole välttämättä enää niitä ajankohtaisimpia. Käytettävyyden periaatteet ovat onneksi pysyneet aika lailla samoina aina 1930-luvun Gestalt-hahmolaeista 90-luvun Nielseniin ja edelleen tälle vuosituhannelle.

Ohjaajani ilmaisivat alusta asti huolensa siitä, että työn aihe on liian laaja ja rönsyilevä, ja sellaiseksi se lopulta jäikin. Kiitokset ja pahoittelut teille. Lähdin tekemään tätä pitkälti sillä asenteella, että pääsen kirjoittamaan itseäni kiinnostavista asioista, mistä seurasi haittapuolena se, että tutkielma ei ole tieteellisesti erityisen merkittävä eikä paneudu kaikkiin aiheisiin kovin syvällisesti. Toivon kuitenkin, että muut aiheesta kiinnostuneet pitävät sen lukemisesta.

Geometrix Oy:lle kiitos siitä, että sain luvan lähteä työstämään tätä innostavaa projektia. Kollegoilleni erityiskiitos siitä, että he mielihyvin jättävät selainpuolen ongelmat minun huolekseni niin, että sain hyvin vapaat kädet kehittää prototyyppiä.

Kiitokset isälleni, jolta voi kysyä mitä tahansa ja saada yleensä oikean vastauksen.

Lopulta haluan kiittää puolisoani siitä, että oli tukemassa työntekoa alusta loppuun. Gradu valmistui sen takia tai siitä huolimatta.

Kehä III:n ulkopuolella, keväällä 2019

Sisältö

1 Johdanto

Käyttöliittymäsuunnittelu on ohjelmistoalan haastavimpia ongelmia. Siinä missä verkkosovelluksen palvelinpuolen on tuettava ohjelmistokomponenttien keskeistä vuorovaikutusta, selainpuolella on lisäksi huomioitava käyttäjän ja koneen välinen vuorovaikutus ja ihmismielen monimutkaiset kognitiiviset prosessit. Lisäksi käyttöliittymän on toimittava sujuvasti näyttökoosta, laitteesta ja selaimesta riippumatta.

Tämän pro gradu -työn tavoitteena on tutkia käytettävyyttä sekä modernien verkkosovellusten käyttöliittymäsuunnittelua erityisesti sellaisissa tapauksissa, joissa sovelluksen on pystyttävä sujuvasti esittämään suuri määrä hallinnoitavaa tietoa. Tutkielman tuloksena syntyy interaktiivinen käyttöliittymän prototyyppi. Työ tehdään toimeksiantona ohjelmistoalan yritykselle, jolla on motivaationa uudistaa olemassa olevan työaikakirjausten katselu- ja hallintasovelluksen käyttöliittymää. Työajanseurannan sovelluspaketti on tarkoitus jatkossa ottaa käyttöön tuhansille uusille käyttäjille, mutta nykyinen käyttöliittymä ei skaalaudu tällaiselle tietomäärälle, sillä se on suunniteltu alun perin korkeintaan muutamien kymmenten työntekijöiden tietojen näyttämiseen. Tarpeena on myös päivittää sovellus mukailemaan tuoteperheen muiden sovellusten ulkoasua ja hyödyntämään modernimpia teknologioita.

Tutkielmassa vastataan seuraaviin kysymyksiin:

- Miten käytettävyys on huomioitava toimistokäyttöön tarkoitetun verkkosovelluksen käyttöliittymän suunnittelussa ja toteutuksessa?
- Mitä parannuksia uusi käyttöliittymäratkaisu tuo verrattuna vanhaan?

Työn tutkimusmenetelmänä on suunnittelutiede (engl. Design Science), jossa tavoitteena on ratkaista organisaatiossa havaittu ongelma kehittämällä ja arvioimalla jokin IT-artefakti [HMPR04], joka on tässä tapauksessa käyttöliittymäprototyyppi. Toteutusosiota pohjustetaan teoriaosuudella, joka on pienehkö kirjallisuustutkimus.

Yhtenä tutkielman haasteena on lähdemateriaalin löytäminen ja hyödyntäminen, sillä esimerkiksi JavaScript-teknologioista on vertaisarvioitua kirjallisuutta hyvin vähän. Suuri osa lähteistä on ohjelmistoalan toimijoiden blogikirjoituksia ja verkkoartikkeleita, joihin on suhtauduttava kriittisesti johtuen verkon alhaisesta julkaisukynnyksestä ja vertaisarvioinnin puutteesta. Kirjoittajien mahdolliset kytkökset heidän arvioimiinsa sovelluskehyksiin on huomioitava. Kaikissa web-suunnittelua koskevissa lähteissä on myös otettava huomioon julkaisuaika ja se, onko niiden sisältö edelleen ajankohtaista nykypäivän ohjelmistokehityksessä.

Haasteena on myös se, miten valmiin prototyypin arviointi toteutetaan. Esimerkiksi kattava käytettävyystestaus vaatii aikaa, resursseja sekä halukkaita testikäyttäjiä, jotka ovat mielellään sovelluksen käyttäjäkohderyhmää. Aika ja resurssit voivat muodostua ongelmaksi myös prototyypin kehitysvaiheessa, mikäli yrityksen muut, kiireisemmät projektit menevät tämän työn toteutuksen edelle.

Tutkielman teoriaosuus käsittää luvut 2 ja 3. Luvussa 2 tutkitaan ihmisen ja koneen vuorovaikutusta sekä käyttöliittymäsuunnittelun perusteita kognitiotieteen käsitteitä hyödyntäen. Toisessa teorialuvussa käsitellään toteutettavan prototyypin kannalta oleelliset web-käyttöliittymien ja käytettävyyden ominaisuudet. Lähteinä käytetään sekä ohjelmistoalan että kognitiotieteen kirjallisuutta ja erilaisia verkkolähteitä. Luvussa 4 määritellään nykyisen käyttöliittymän ongelmat ja uuden toteutuksen vaatimukset sekä tehdään alustava käyttöliittymäsuunnitelma. Luvussa 5 valitaan kehitystyökalut, kuvataan varsinaisen prototyypin toteutus ja perustellaan siinä tehdyt valinnat. Lopuksi luvussa 6 arvioidaan prototyyppi ja luvussa 7 tehdään yhteenveto työstä, sen onnistumisesta sekä mahdollisesta jatkokehityksestä.

2 Kognitiotiede käyttöliittymäsuunnittelussa

Mitä vähemmän aikaa käyttäjällä menee käyttöliittymän sisäistämiseen, sitä tehokkaampana käyttöliittymää voidaan pitää. Hyvä käytettävyys saavutetaan huomioimalla loppukäyttäjän näkökulma palvelua suunnitellessa. Käytettävyyssuunnittelussa on hyötyä kognitiotieteestä, joka tutkii muun muassa havaitsemista, oppimista ja päätöksentekoa. Helppokäyttöisten ja toimivien käyttöliittymien kehittäminen vaatii ymmärrystä siitä, miten käyttäjät – eli ihmiset – havaitsevat ja ymmärtävät ruudulla näkyvät asiat. Tässä luvussa käydään läpi käyttöliittymäsuunnittelun kannalta oleellisia kognitiotieteen käsitteitä.

2.1 Mentaaliset mallit

Toistuva altistuminen jollekin tilanteelle rakentaa mieleen kuvauksen siitä, mitä tilanteelta voi odottaa. Tilanteen mentaalinen malli sisältää tiedon objekteista ja tapahtumista, jotka ovat kyseiselle tilanteelle tyypillisiä. Nämä mallit helpottavat elämää vähentämällä tarvetta huomioida kaikkia ympäristön yksityiskohtia, mutta ne voivat myös vääristää havaintoja. [Joh14]

Kokeneemmilla verkkosovellusten käyttäjillä on mentaalinen malli siitä, millainen tyypillinen web-sovellus on ja missä esimerkiksi sovelluksen nimen, navigaation tai hakutoiminnon pitäisi sijaita. Voidaan myös puhua transferenssista eli siirtovaikutuksesta, jolla viitataan tässä yhteydessä käyttäjän aiempiin kokemuksiin perustuviin odotuksiin [YLYZ12]. Mentaaliset mallit nopeuttavat ja tehostavat käyttäjän ja järjestelmän välistä vuorovaikutusta [GB16]. Malleihin luottavat käyttäjät voivat klikata painikkeita ja linkkejä katsomatta niitä tarkasti, sillä heitä ohjaavat enemmän omat tottumukset kuin se, mitä ruudulla todella näkyy [Joh14].

Mentaalinen malli järjestelmän toiminnasta muodostuu toistuvassa prosessissa, jossa on seitsemän vaihetta: tavoitteen muodostaminen, aikomuksen muodostaminen, toiminnon valinta, toiminnon toteuttaminen, tilan havaitseminen, tilan tulkitseminen ja lopputuloksen arviointi [GB16]. Järjestelmän käyttäjällä on tyypillisesti jokin tavoite, joka suunnittelijan tulisi ymmärtää. Sovellusta suunnitellessa on varmistettava, että käyttäjällä on aina saatavilla tieto, joka mahdollistaa tavoitteen saavuttamisen. Tavoite ohjaa aistijärjestelmiä ja suodattaa pois sellaisia havaintoja, jotka eivät liity kyseiseen tavoitteeseen. Esimerkiksi web-sovelluksessa navigoiva käyttäjä usein jättää kokonaan huomioimatta tiedon, joka ei vaikuta liittyvän hänen tavoitteeseensa [Joh14]. Toiminnon suoritettuaan käyttäjä arvioi lopputulosta

järjestelmän antaman vastauksen perusteella, mikä mahdollistaa mentaalisen mallin kehittymisen ja hiomisen [GB16].

Monimerkityksellisiä näkymiä on syytä välttää, ja käyttöliittymäsuunnitelmaa arvioidessa on varmistettava, että kaikki käyttäjät ymmärtävät näkymän samalla tavalla. Esimerkiksi painikkeiden alalaitaan lisätään usein varjostusta luomaan vaikutelmaa siitä, että kyseinen komponentti on koholla suhteessa ympäristöönsä. Tässä käytännössä luotetaan siihen yleiseen käsitykseen, että valonlähde on näytön yläreunassa. Mikäli varjo esitettäisiin jossain toisessa paikassa, ei käyttäjälle syntyisi samanlaista mielikuvaa. [Joh14]

2.2 Tottuminen

Havaintojärjestelmän herkkyys havainnolle vähenee, kun altistuminen samalle tai usealle samankaltaiselle havainnolle on toistuvaa [Joh14]. Tämä on otettava huomioon silloin, kun halutaan suunnitella käyttöliittymään erityisen erottuvia elementtejä. Esimerkiksi animaatiot ja kirkkaat värit toimivat huomion herättämisessä, mutta niiden jatkuva käyttö saa käyttäjän tottumaan niihin tyypillisenä ympäristön osana, jolloin ne eivät enää ohjaa huomiota puoleensa [Eva17]. Toinen esimerkki tottumisesta on käyttäjien reagointi toistuviin virhe- tai varmistusviesteihin verkkosivuilla; alkuun nämä saattavat kiinnittää käyttäjän huomion, mutta useaan kertaan toistuneet varmistusviestit usein suljetaan refleksinomaisesti viestiä lukematta [Joh14].

Sovelluksen tietosisällön ja kontrollien sijoittelun kannattaa olla johdonmukaista. Samaa funktiota palvelevien komponenttien pitäisi eri näkymissä sijaita samassa kohdassa ja näyttää samalta. Yhdenmukaisuus helpottaa ja nopeuttaa komponenttien löytämistä ja ymmärtämistä. Jos esimerkiksi ”Edellinen”- ja ”Seuraava”-painikkeiden paikkaa yllättäen vaihdetaan monivaiheisen lomakkeen viimeisellä sivulla, suurin osa käyttäjistä ei heti huomaa muutosta, sillä näköjärjestelmä on ehtinyt tottua johdonmukaiseen sijoitteluun eikä ole valppaana muutoksille. Käyttäjät ovat tottuneet etsimään tutun näköisiä elementtejä tutuista paikoista, jolloin elementti voi jäädä huomaamatta, mikäli se on epätyypillisen näköinen tai epätyypillisessä sijainnissa. Mikäli yksi sovelluksen lukuisista ”Tallenna”-painikkeista onkin poikkeuksellisen värinen tai muotoinen, eivät käyttäjät välttämättä löydä sitä. [Joh14]

2.3 Asettelu ja värit

Käyttäjä havainnoi verkkosivuja tyypillisesti F-kirjaimen muotoisella kaavalla, jossa ensimmäiseksi luetaan sivun ylälaitaa vaakasuoraan [Nie06]. Tämän jälkeen huomio siirtyy hieman alaspäin ja etenee uudestaan vaakasuoraan F-kirjaimen alemman vaakaviivan mukaisesti. Lopuksi käyttäjän katse etenee edelleen alaspäin sivun vasenta laitaa pitkin. Näin ollen sivun oikea alanurkka jää yleensä vähimmälle huomiolle [Eva17]. F-kaavan huomioivassa käyttöliittymäsuunnittelussa tähän nurkkaan ei sijoiteta mitään tärkeää sisältöä ja esimerkiksi navigaatiolinkit asetellaan sivun yläosaan tai vasempaan reunaan. Ulkoasun tilankäytön kannalta ylhäällä sijaitseva navigaatio on parempi vaihtoehto, mutta toisaalta sijoittelu vasempaan reunaan tukee navigaation vertikaalista selaamissuuntaa, joka on käyttäjille luonnollisempi vaihtoehto. Yläreunan navigaatio on verkossa yleisimmin käytössä mukautuvuutensa takia [Cre13].

Ihmisen näkökenttä jakautuu aivopuoliskojen mukaan vasempaan ja oikeaan näkökenttään, joista oikea puoli on vastuussa sanojen ja tekstin lukemisesta ja vasen puolestaan on erikoistunut kuvien tulkintaan. Käyttäjien kognitiivista taakkaa voidaan pienentää asettelemalla teksti ja kuvat näiden näkökenttien mukaan. Kognitiivista taakkaa voidaan helpottaa myös rajaamalla käyttöliittymässä kerrallaan näytettävien asioiden määrää; useimpien ihmisten aivojen kapasiteetti riittää 5–9 asian, esimerkiksi numeron, säilyttämiseen työmuistissa. [YLYZ12, QF16]

Värejä käytetään usein huomion herättämiseen ja yleisesti helpottamaan käyttöliittymän ymmärtämistä. Kirkkaiden värien käyttämistä on syytä välttää, sillä ne saavat silmän pupillin supistumaan ja aiheuttavat näin lihasväsymystä. Erityisen paljon silmää rasittaa vaaleiden ja kirkkaiden sävyjen yhdistely, sillä tällöin pupilli joutuu vuorotellen laajenemaan ja supistumaan. Vihreän ja punaisen värin käyttämistä kannattaa myös välttää, sillä niiden erottaminen tuottaa vaikeuksia värisokeille. [YLYZ12]

2.4 Ikonit ja symboliikka

Sovelluskehityksessä käytettävät ikonit ovat symboleita, joiden tarkoitus on välittää tietoa ja jotka ovat helposti tunnistettavia ja muistettavia. Ikonien suunnittelussa oleellista on yksinkertaisuus ja yhdenmukaisuus. Ikoni on intuitiivisempi ja visuaalisesti miellyttävämpi ilmaisutapa kuin teksti, mutta se ei pysty tarjoamaan samaa tarkkuutta informaation välittämisessä. Ikonisuunnittelun periaatteista tärkein on

ikonin tunnistettavuus: käyttäjän pitäisi heti ymmärtää ikonista ja sen kontekstista, mitä se tarkoittaa. Käytännöllisyys on oleellisempaa kuin visuaalinen kauneus, mistä johtuen ei ole suositeltavaa pyrkiä liian omaperäiseen ja taiteelliseen ilmaisuun, vaan kannattaa mukailla yleisesti käytettyjä ja tunnistettavia tyylejä. Jos sivulla on useita ikoneita, on niiden erotuttava selkeästi toisistaan. [QF16]

Käyttäjien keskuudessa suosituimpia ikoneita ovat ne, joissa on joko musta kohde – esimerkiksi musta kuvio valkoisella tai keltaisella taustalla – tai musta tausta. Valkoiset kohteet ja taustat olivat toiseksi suosituimpia. Kromaattisista väriyhdistelmistä suosituin oli keltainen väri sinisellä taustalla. Lisäksi väriyhdistelmä arvioidaan paremmaksi sen esiintyessä yksinään kuin usean muun saman väriyhdistelmän elementin kanssa yhtäaikaisesti. [QF16]

Qiang ja Fei [QF16] esittävät ikonisuunnittelun prosessin, jossa on neljä ulottuvuutta: elementtien semanttinen ilmaisu, elementtien jäsentely, käyttöliittymän ja kulttuurin konteksti ja käyttäjän kognitiiviset piirteet. Toimivan ikonin ehtona on, että käyttäjä ymmärtää sen tarkoituksen mielleyhtymien avulla; esimerkiksi asiakaspalvelua kuvastavaan ikoniin voitaisiin valita kuva henkilöstä, jolla on kuulokemikrofoni päässä. Toisella tasolla määritellään ikonin rakenne, mihin sisältyvät muun muassa elementtien mittasuhteet ja asettelu. Käyttöliittymän ja kulttuurin kontekstilla tarkoitetaan ikonin graafista ympäristöä sekä käyttäjän kulttuuritaustaa ja ympäristöä, jotka molemmat vaikuttavat ikonin ymmärtämiseen ja on otettava huomioon sen suunnittelussa. Prosessin neljännessä ulottuvuudessa hyödynnetään ymmärrystä käyttäjän kognitiivisista prosesseista ja esimerkiksi mentaalisista malleista, jotta ikonista saadaan mahdollisimman asiaankuuluva ja ymmärrettävä.

2.5 Hahmopsykologia

Hahmopsykologia eli Gestalt-teoria on psykologian suuntaus, jonka mukaan havaittu kokonaisuus on muuta kuin vain osiensa summa [Kof35]. Tätä havainnollistetaan hahmolaeilla, jotka kuvaavat sitä, miten havaitut kohteet käsitetään laajempina kokonaisuuksina. Esimerkiksi osittain piilossa olevan objektin koetaan jatkuvan peittävän pinnan takana, mikäli objektin näkyvät osat ovat keskenään riittävän yhteneviä värin, tekstuurin, muodon ja liikkeen suhteen. Kun hahmopsykologiaa sovelletaan digitaaliseen mediaan [Eva17], voidaan päätellä käyttäjän kokevan elementit toisiinsa liittyviksi, mikäli jokin seuraavista ehdoista täyttyy:

- elementit ovat lähellä toisiaan

- elementeillä on samankaltainen väri tai sävy
- elementeillä on samankaltainen koko tai muoto
- elementit liikkuvat samaan suuntaan
- elementit sijaitsevat samalla janalla
- elementit ovat saman alueen sisällä
- elementit ovat kiinni toisissaan.

Hahmopsykologian periaatteiden avulla käyttöliittymän elementtien tarkoitus on mahdollista ymmärtää ilman erillistä ohjeistusta. Läheisyyden hahmolaki on oleellinen erityisesti sovellusten lomakkeiden ja ohjauspaneelien suunnittelussa, jolloin toisiinsa liittyvät painikkeet kannattaa sijoittaa lähelle toisiaan. Tällainen lähestymistapa tekee käyttöliittymästä siistimmän ja vähentää tarvittavan koodin määrää verrattuna siihen, että painikkeet ryhmiteltäisiin esimerkiksi laatikoiden tai erottimien avulla. Hahmolakien hyödyntäminen helpottaa käyttäjän huomion ohjaamista ja tekee käyttöliittymästä intuitiivisemman käyttää. [Eva17]

3 Käyttöliittymäsuunnittelu verkkosovelluksissa

Verkkosovellukset eroavat verkkosivuista siinä, että niissä on dynaamista sisältöä ja interaktiivista toiminnallisuutta. Useat työpöytäsovellukset siirretään nykyään verkkoon, mikä helpottaa sovelluksen ylläpitoa ja mahdollistaa palvelun käytön käyttäjän laitteesta riippumatta [KKGF16]. Käyttöliittymien ja käyttökokemuksen suunnittelulle annetaan yhä enemmän painoarvoa, millä halutaan parantaa tuotteen asiakastyytyväisyyttä ja markkinoitavuutta sekä minimoida sovellustuen vaatimat resurssit. Tässä luvussa avataan modernien web-sovellusten käyttöliittymäsuunnittelun kannalta oleellisia termejä.

3.1 Yksisivuiset sovellukset

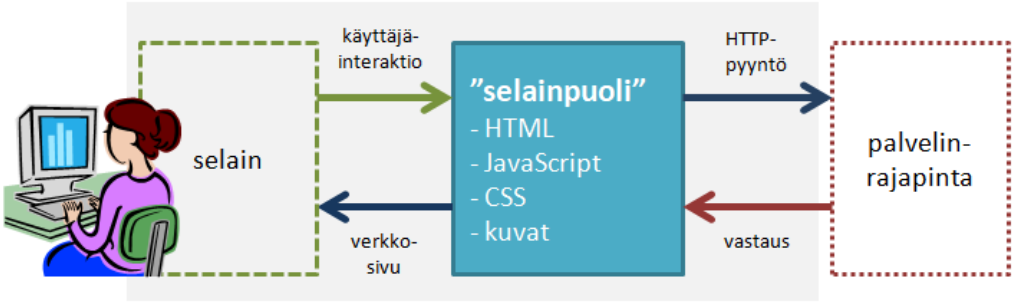
Yksisivuisella sovelluksella (engl. Single-page Application, SPA) tarkoitetaan web-sovellusta, joka koostuu vain yhdestä, päivittyvästä HTML-dokumentista. Kun sovellus on kerran ladattu selaimeen, linkkien painaminen ei vie käyttäjää kokonaan uudelle sivulle, vaan ainoastaan päivittää jotakin osiota sivun sisällä. Perinteisillä verkkosivuilla siirtymä sivulta toiselle synnyttää pyynnön web-palvelimelle, joka tekee tietokantakyselyn ja generoi sen perusteella näytettävän HTML-sivun dynaamisesti. [Leh13]

Yksisivuisessa sovelluksessa HTML-koodi, tyylimäärittelyt ja skriptit ladataan käyttäjän selaimeen, ja palvelimelta pyydetään ainoastaan näytettävä tieto esimerkiksi JSON-muodossa (JavaScript Object Notation). Tämä helpottaa palvelimen kuormaa, sillä se joutuu ainoastaan lähettämään selainpuolen pyytämää raakadataa eikä osallistu lopullisten näkymien rakentamiseen [Ruf14]. Vasteajat ovatkin yksisivuisissa sovelluksissa huomattavasti parempia kuin perinteisissä verkkopalveluissa, mutta sivun ensimmäinen lataus voi viedä enemmän aikaa [fus18]. Käyttäjän kannalta yksisivuinen sovellus tarjoaa nopeampaa ja aidomman tuntuista vuorovaikutusta sekä helpommin ymmärrettävän käyttökokemuksen, sillä käyttäjän toimenpiteet eivät ikinä aiheuta koko sivun kontekstin vaihtumista kuten perinteisillä verkkosivustoilla [Leh13].

Tyypillinen yksisivuinen sovellus mukailee MVC-arkkitehtuuria, jossa rakenne jaetaan malliin (engl. Model), näkymään (engl. View) ja käsittelijään tai ohjaimeen (engl. Controller). Näkymä renderöi käyttöliittymän, jota se päivittää ohjaimen kautta mallilta saapuvan tiedon perusteella, ja malli puolestaan vastaa tiedon lukemisesta ja kirjoittamisesta [Sha17]. Tämän jaon tavoitteena on erottaa toisistaan

sovelluksen käsittelemä tieto, tiedon esitystapa ja tietojen käsittely [Leh13]. Käytännössä täysin eriytetty esitystapa tarkoittaa sitä, että sovelluksen käyttämä CSS-tyylimäärittely voidaan vaihtaa toiseen ilman, että mihinkään muuhun sovelluksen osaan tarvitsee koskea. Tämä helpottaa usein vaihtoehtoisten käyttöliittymien tekemistä ja testaamista.

Tilan hallinta on monimutkaista yksisivuisessa sovelluksessa. Näkymä voi muuttua käyttäjän interaktion seurauksena, mallissa oleva arvo voi muuttua aiheuttaen näkymän päivittymisen, koko sovelluksen tila voi muuttua näkymän vaihtuessa toiseen ja palvelinpuolelta voi saapua sisältöä viiveellä. Monimutkaisuutta helpottavat valmiit sovelluskehykset, jotka vastaavat tilojen siirtymistä. [Sha17]



Kuva 1: Esimerkki selaimen ja palvelimen välisestä vuorovaikutuksesta

Yksisivuisten sovellusten varsinaisen selainpuolen toiminnallisuus toteutetaan JavaScriptillä, joka on ainoa selainten yleisesti tukema ohjelmointikieli. Käytännössä JavaScriptin tehtävänä on päivittää selaimen näkymää lisäämällä, poistamalla ja muokkaamalla käyttöliittymän elementtejä, tarkkailla käyttäjältä saatuja syötteitä ja lähettää verkkopyyntöjä palvelimelle. [Leh13]

Tiedonsiirto selain- ja palvelinpuolen välillä tapahtuu Ajax-tekniikalla. Tämä tarkoittaa sitä, että tiedonsiirto suoritetaan asynkronisesti eli eriaikaisesti, jolloin palvelimelle tehty pyyntö ei estä käyttäjää jatkamasta sovelluksen käyttöä vastausta odotellessa [Gar05]. Koska vastaus asynkroniseen pyyntöön saadaan tuntemattoman ajan päästä, mutta käyttäjä kuitenkin odottaa välitöntä palautetta toiminnoistaan, joudutaan käyttöliittymässä usein esittämään jonkinlainen viesti, joka kertoo toiminnon onnistuneen, vaikka vastaus palvelimelta ei ole vielä saapunut.

Selaimen ja palvelimen välistä vuorovaikutusta sekä tämän työn rajausta on havainnollistettu kuvassa 1. Keskimmäisessä laatikossa on esitettynä se osuus sovelluksesta, joka syntyy tämän tutkielman lopputuloksena. Toteutukseen sisältyy myös selaimen

ja palvelimen välisen tiedonsiirron suunnittelu sekä käyttäjän vuorovaikutuksen ja selaintoimivuuden huomiointi. Työaikakirjauksen tiietojen syöttäminen lomakkeelle on esimerkki käyttäjäinteraktiosta, joka tapahtuu selainpäässä. Kun käyttäjä tallentaa lomakkeen, lähettää selaimen JavaScript HTTP-päivityspyynnön palvelimelle. Vastaanotettuaan ja käsiteltyään pyynnön palvelin palauttaa selaimelle vastauksen, joka voi olla kuittaus onnistuneesta tallennuksesta tai esimerkiksi virheviesti.

3.2 Responsiivinen suunnittelu

Erilaisten laitteiden ja näyttökokojen määrä on tuonut oman haasteensa verkkosivujen suunnitteluun, sillä saman sovelluksen pitäisi tarjota samanlainen käyttökokemus useilla eri laitteilla. Usean eri laitteille optimoidun sovelluksen ylläpitäminen tuottaisi huomattavasti lisätyötä kehittäjille [LHH14]. Responsiivinen suunnittelu tarjoaa ratkaisun tähän ongelmaan. Responsiivisuus tarkoittaa teknisellä tasolla pääasiassa kolmea asiaa: joustavaa ruudukkopohjaista ulkoasua (engl. grid layout), joustavaa mediaa kuten kuvia ja videoita sekä CSS3-tyylimäärittelyjä, jotka muokkaavat sivua käyttäjän näyttökoon mukaan [Cre13, LHH14, BB13]. Ruudukkopohjaisella ulkoasulla tarkoitetaan käyttöliittymän jakamista sarakkeisiin, jotka mukautuvat sivun leveyden mukaan. Responsiivisen suunnittelun tavoitteena on muun muassa se, että käyttäjä pystyy selaimen koosta riippumatta näkemään sivun koko sisällön ilman horisontaalista vierittämistä [LHH14]. Mikäli on tarpeen, voidaan esimerkiksi mobiilinäkymästä karsia kokonaan pois kuvia tai muuta sisältöä, joka halutaan näyttää vain sovelluksen työpöytäversiossa [BB13].

Responsiivisessa suunnittelussa on kaksi lähestymistapaa: työpöytälähtöinen ja mobiililähtöinen (engl. desktop first, mobile first). Mobiililähtöisessä lähestymistavassa ulkoasu suunnitellaan oletuksena pienemmälle näyttökoolle sopivaksi niin, että käyttöliittymän komponentit laajenevat resoluution kasvaessa. Työpöytälähtöisessä suunnittelussa lähtökohtana on työpöytänäkymä, joka vastaavasti muuttuu yksinkertaisemmaksi resoluution pienentyessä. [Moh13]

3.3 Vasteajat ja suorituskyky

Käyttäjä kokee järjestelmän reagoivan interaktioon välittömästi, kun vastauksen odottamiseen menevä aika pysyy noin 0,1 sekunnissa. 0,1–1,0 sekunnin kohdalla käyttäjä huomaa jo viiveen, ja yli sekunnin vasteaika vaatii jonkinlaisen palautteen ja indikaattorin siitä, että prosessi on käynnissä. Prosessin etenemistä kuvaava indi-

kaattori viestii käyttäjälle, että järjestelmä ei ole kaatunut, ja tarjoaa lisäksi jotakin katseltavaa, mikä tekee odottamisesta vähemmän turhauttavaa. Jälkimmäisen syyn takia esimerkiksi graafinen etenemispalkki (engl. progress bar) on tekstiä suositeltavampi ratkaisu, sillä se on visuaalisesti mielenkiintoinen. Yli 10 sekuntia kestävissä prosesseissa käyttäjä tarvitsee lisäksi palautetta siitä, milloin järjestelmä arvioi tehtävän valmistuvan. Suhteellisen lyhyillä, alle 10 sekunnin odotusajoilla ratkaisuksi riittää indikaattori, joka kertoo järjestelmän olevan "kiireinen" muttei ota kantaa jäljellä olevaan aikaan. Myös se vaihtoehto on huomioitava, että prosessi valmistuukin niin nopeasti, ettei käyttäjä ehdi edes huomata indikaattoria. [Nie94]

Verkkosovellusten latausajoista suuri osa menee selainpuolen prosessointiin sekä erilaisten resurssien kuten tyylimäärittelyjen, skriptien ja kuvien lataamiseen. Resurssien lataamisesta aiheutuvia latausaikoja voi pienentää vähentämällä sovelluksen tekemiä verkkopyyntöjä eli yhdistämällä esimerkiksi useita eri skriptejä samaan tiedostoon. Ihannetilanteessa verkkosovellus käyttää tuotannossa kokonaisuudessaan vain yhtä CSS-tiedostoa ja yhtä JavaScript-tiedostoa. Tiedostot voi lisäksi tiivistää niin, että niistä poistetaan kaikki ylimääräinen sisältö, kuten välilyönnit, sisennykset ja kommentit, jolloin tiedostokoko pienenee. [Cre13]

Vasteaikoja on mahdollista parantaa valitsemalla sovellukselle sopivin tapa tietosisällön hallintaan. Yksi vaihtoehto on ladata palvelimelta sisältöä sitä mukaa, kun sitä tarvitaan. Tämä mahdollistaa sovelluksen nopean alustuksen, mutta voi myöhemmin näkyä pitkinä vasteaikoina toimintoja suorittaessa. Päinvastaisena vaihtoehtona on ladata kaikki sovelluksen tarvitsema sisältö saman tien selaimeen, jolloin sovelluksen käynnistymisessä voi mennä aikaa, mutta myöhempi käyttö puolestaan on sujuvaa. Sisällön lataaminen alustuksen yhteydessä voidaan toteuttaa myös niin, että lataaminen tapahtuu taustalla eikä estä sovelluksen käyttöä niissä osioissa, joiden tarvitsema sisältö on jo ehditty ladata. [Leh13]

Dokumenttioliomalli eli DOM on selaimen muistissa oleva HTML-merkinnän representaatio, joka tarjoaa rajapinnan sen alkioiden eli HTML-elementtien käsittelyyn esimerkiksi JavaScriptillä. HTML-dokumenttioliomalli noudattaa puumaista rakennetta, mistä johtuen alkioihin pääsy voi viedä paljon aikaa sivuilla, joilla elementtejä on runsaasti [Dac17]. Dynaamiset web-sovellukset päivittävät alkioita jatkuvasti, jolloin HTML-dokumenttioliomallista aiheutuu huomattavia suorituskykyongelmia. Lisäksi monet JavaScript-sovelluskehykset päivittävät DOMia enemmän kuin olisi tarpeen; esimerkiksi yhtä listan elementtiä päivittäessä sovelluskehys saattaa rakentaa koko listan uudestaan. Ratkaisuksi tähän on kehitetty virtuaalinen dokument-

tioliomalli, joka on abstraktio HTML-dokumenttioliomallista [Kra15]. Virtuaalista dokumenttioliomallia käsitellään uudestaan alaluvussa 5.2.

3.4 Tietomassan esittäminen

Suuren tietomäärän listamaiseen esittämiseen on kaksi yleistä ratkaisua: tiedon jakaminen usealle sivulle ja loputon vieritys (engl. infinite scrolling), jossa sisältöä ladataan jatkuvasti lisää käyttäjän selatessa sivua alaspäin [Pra17, Pie15, Bab16].

Loputon vieritys toimii parhaiten sosiaalisessa mediassa, jossa tavoitteena on saada käyttäjä pysymään sivulla mahdollisimman pitkään tarjoamalla jatkuvasti uutta sisältöä [Bab16, Pra17]. Käyttäjä voi hämmentyä jatkuvasti lisääntyvästä tietomäärästä ja kokea menettävänsä kontrollin siitä [Pie15]. Tietyn tiedon löytäminen uudestaan sivulle myöhemmin palattaessa on myös hankalaa. Babichin artikkelin [Bab16] mukaan käyttäjät kuitenkin pitävät sivun vierittämistä käyttökokemuksena miellyttävämpänä kuin klikkailemista. Kosketusnäytöt ja hiirten rullat mahdollistavat helpon ja nopean selaamisen.

Tiedon sivutus mahdollistaa sen, että käyttäjä voi siirtyä nopeasti haluamaansa kohtaan tietomassaa. Sivutetulle tiedolle voidaan myös tarjota mahdollisuudet tiedon järjestämiseen ja suodattamiseen [Pra17]. Lisäksi sivutus tarjoaa loputonta vieritystä paremman kontrollin tunteen käyttäjälle, sillä tietomäärällä on selkeä alku ja loppu [Pie15]. Ratkaisun haittapuolena on se, että tiedon selaaminen vaatii ”Seuraava sivu” -painikkeen löytämisen, painikkeen klikkaamisen ja lopuksi vielä jonkin verran odottamista, kun seuraavan sivun tiedot latautuvat [Bab16]. Ongelma pahenee, jos yhdellä sivulla näytettävän tiedon määrä on liian pieni.

Sekä Babich [Bab16] että Pierce [Pie15] tiivistävät, että loputon vieritys sopii parhaiten käyttäjien luoman ja visuaalisen sisällön päämäärättömään selailuun, kun taas sivutus on parempi vaihtoehto sovelluksiin, joissa käyttäjällä on selkeät toimintatavoitteet.

3.5 Käytettävyys

ISO 9241-11 -standardin määritelmässä käytettävyys mittaa sitä, miten vaikuttavasti, tehokkaasti ja tyydyttävästi (engl. effectiveness, efficiency, satisfaction) määritelty käyttäjäjoukko pystyy hyödyntämään tuotetta saavuttaakseen määritellyt tavoitteet määritellyssä käyttökontekstissa. Vaikuttavuuden määrää se, miten tar-

kasti ja täydellisesti käyttäjä onnistuu pääsemään tavoitteisiinsa. Tehokkuus puolestaan suhteuttaa tavoitteisiin pääsyn käytettyihin resursseihin, kuten aikaan, hintaan tai käyttäjän näkemään fyysiseen ja henkiseen vaivaan. Tyydyttävyys taas viittaa käyttäjien subjektiiviseen tyytyväisyyteen. Käytettävyyden määrittely ja arviointi vaatii tavoitteiden ja käyttökontekstin – eli käyttäjien, tehtävien, tarvikkeiden ja ympäristöjen – kuvaamisen. [iso98]

Nielsen [Nie94] määrittelee käytettävyyden hyödyllisyyden (engl. usefulness) alakäsitteeksi, johon kuuluvat opittavuus, tehokkuus, muistettavuus, virheiden määrä ja tyydyttävyys. Järjestelmän tulisi olla nopeasti opittava, jotta käyttäjä pääsee mahdollisimman pian tekemään sillä tuottavaa työtä. Tehokkuus viittaa tuottavuuteen, joka kokeneiden käyttäjien on mahdollista saavuttaa. Järjestelmän pitäisi myös olla helposti muistettava niin, että käyttäjä osaa käyttää sitä myös käyttötauon jälkeen. Käyttäjälähtöisten virheiden mahdollisuuden pitäisi olla minimoitu, ja virheiden pitäisi olla helposti korjattavissa. Lisäksi järjestelmän käyttökokemuksen pitäisi olla miellyttävä.

Omaksi termikseen voidaan erottaa käyttökokemus (engl. user experience), joka kattaa käytettävyyttä laajemmin käyttäjän ja järjestelmän välisen interaktion, mukaan lukien vuorovaikutuksesta syntyvät ajatukset, tuntemukset ja havainnot [TA13].

Käytettävyyden merkitys korostuu järjestelmissä, joissa virheet saattavat johtaa omaisuuden vaurioitumiseen, taloudellisiin menetyksiin, ihmisten loukkaantumiseen tai jopa kuolemaan [TA13].

3.6 Määrittely ja suunnittelu

Sovelluksen vaatimusmäärittelyssä tulisi vastata muun muassa seuraaviin kysymyksiin [Leh13]:

- Mikä on sovelluksen tarkoitus?
- Mitä toimintoja sovellukseen sisältyy?
- Millä laitteilla ja selaimilla sovelluksen tulee toimia, ja mitkä käyttöympäristöt ovat ensisijaisia?

Sovellusratkaisun hahmotteluvaiheessa käyttöliittymän kannalta oleellista on suunnitella selain- ja palvelinpuolen väliset rajapinnat ja niiden välisten viestien muoto.

Hahmotelman pohjalta voidaan luoda valituilla JavaScript-työkaluilla karkean tason prototyyppi, jonka tarkoituksena on toimia toteutettavan sovelluksen runkona ja mahdollisesti myös ensimmäisenä esittely- ja testauskelpoisena versiona lopputuotteesta. Lisäksi käyttöliittymästä laaditaan ensimmäinen luonnos eli rautalankamalli, joka määrittää käyttöliittymän elementtien sijoittelun. Rautalankamallin perusteella tehdään tarkempi graafinen suunnitelma. [Leh13]

Käyttäjäkeskeisessä suunnittelussa oleellista on keskittyä alusta asti kohdekäyttäjäryhmän käyttötarpeisiin ja ottaa käyttäjät mukaan suunnitteluprosessin ideointiin ja arviointiin. Vaatimusten pohjalta voidaan laatia vaihtoehtoisia käyttöliittymäsuunnitelmia, joista kehitetään interaktiiviset prototyypit. Kehitystyökalut tulisi valita siten, että prototyyppien luominen ja muokkaaminen on nopeaa. [GB16]

3.7 Käytettävyystestaus ja arviointi

Testausvaiheessa valmiin sovelluksen laatua voidaan arvioida seuraavilla kysymyksillä [Leh13]:

- Toteuttaako sovellus kaiken määritellyn toiminnallisuuden?
- Toimiiko sovellus virheettömästi kaikissa tilanteissa?
- Onko sovelluksen käyttö mahdollisimman helppoa ja miellyttävää?

Sovelluksen ylläpidon ja jatkokehityksen kannalta on oleellista, että sovelluksen rakenne ja toteutus ovat niin selkeitä, että muutkin kuin alkuperäiseen kehitykseen osallistuneet kehittäjät voivat työskennellä sen parissa [Leh13].

Käytettävyyttä arvioidaan tyypillisesti valvotuilla käytettävyystesteillä, joissa kohdekäyttäjäryhmää edustavalle joukolle testikäyttäjiä annetaan suoritettavaksi joukko määrättyjä tehtäviä sovelluksessa. Testaustilanteessa on läsnä osallistujan lisäksi käytettävyysasiantuntija, joka toimii testauksen vetäjänä. Osallistujaa kehotetaan ajattelemaan ääneen tehtäviä suorittaessaan, ja vetäjä huolehtii mahdollisiin kysymyksiin vastaamisesta ja nauhoittaa istunnon. Toisessa, ei-valvotussa menetelmässä testaaminen tapahtuu verkossa, mikä mahdollistaa datan keräämisen useilta, mahdollisesti maantieteellisesti hajallaan olevilta osallistujilta lyhyen ajan sisällä. [TA13]

Järjestelmän opittavuutta voidaan arvioida laskemalla aika, joka ensimmäistä kertaa sovellusta käyttävillä testihenkilöillä menee annettujen tehtävien suorittamiseen

onnistuneesti. Tehokkuuden arviointiin puolestaan tarvitaan kokeneempia testihenkilöitä, jotka ovat käyttäneet järjestelmää määritellyn ajan – esimerkiksi vuoden tai uuden järjestelmän tapauksessa joitakin tunteja – ennen testaustilannetta. Kolmantena kategoriana ovat satunnaiskäyttäjät, joilla on sovelluksesta jonkin verran aikaisempaa käyttökokemusta ja joiden avulla voidaan arvioida järjestelmän muistettavuutta. [Nie94]

Käyttäjien tekemien virheiden määrä toimii myös mittarina käytettävyysarvioinnissa Virheeksi lasketaan sellainen toiminto, joka ei johda haluttuun lopputulokseen. Jotkin virheet käyttäjä huomaa ja korjaa heti, ja ne tulevat ilmi tehokkuuden arvioinnissa, joten niitä ei tarvitse huomioida erikseen. Vakavammat virheet ovat sellaisia, jotka aiheuttavat esimerkiksi käyttäjän työn katoamisen tai joita käyttäjä ei huomaa, mikä johtaa virheelliseen lopputulokseen. [Nie94]

Nielsenin viides käytettävyyden osa-alue, tyydyttävyys, kuvaa käyttäjän kokemaa subjektiivista mielihyvää järjestelmän käytöstä. Tämä ominaisuus on tärkeämpi vapaa-ajalla käytettävissä ja viihteellisissä kuin toimistokäyttöön tarkoitetuissa sovelluksissa. Käyttäjätyytyväisyyden arviointiin voidaan käyttää psykofysiologisia mittareita, kuten verenpainetta ja ihon joustavuutta, tai yksinkertaista kyselytutkimusta [Nie94]. ISO 9241-11 -standardin mukaan tyydyttävyyden arviointimenetelmiin kuuluu myös esimerkiksi käyttäjän positiivisten ja negatiivisten kommenttien kirjaaminen käytön yhteydessä [iso98].

Riittävä osallistujamäärä käytettävyystestauksessa on viisi henkilöä, sillä viiden testaajan avulla on löydettävissä lähes yhtä monta käytettävyysongelmaa kuin huomattavasti suuremmalla joukolla. Ketterissä, pienen budjetin projekteissa vieläkin pienempi määrä voi olla ihanteellinen. [Nie12]

3.8 Käytettävyysongelmien automaattinen havaitseminen

Käytettävyystestaus mahdollistaa oikeiden käyttäjien käyttötietojen ja kokemusten keräämisen. Haittapuolena on kuitenkin se, että testikäyttäjien rekrytointi, testien suunnittelu, tulosten analysointi sekä ongelmakohtien löytäminen ja ratkaiseminen vaativat aikaa ja resursseja. Manuaalisen työn helpottamiseksi on olemassa erilaisia automatisoituja ratkaisuja etänä tehtävään käytettävyystestaukseen, jossa käyttäjän interaktiot kirjataan lokiin ja analysoidaan, mutta tämä vaatii edelleen paljon tekemistä käytettävyysasiantuntijalta [GGRR17].

Grigera ym. esittelevät kehittämänsä Usability Smells Finder -työkalun, joka tun-

nistaa käytettävyysongelmat automaattisesti käyttäjäinteraktiota analysoimalla ja antaa myös konkreettisia ehdotuksia puutteiden korjaamiseksi [GGRR17]. Taulukkoon 1 on koottu artikkelissa esitetyistä ongelmista ne, jotka on todettu tämän työn tapaustutkimuksen kannalta oleellisiksi. Vaikka tämän työn puitteissa ei päädyttäisikään etänä tehtävään käytettävyystestaukseen, taulukko huomioidaan toteutus- ja arviointivaiheessa.

Ongelman kuvaus	**Ratkaisuehdotus**
Käyttäjät eivät ymmärrä elementin tarkoitusta ja yrittävät saada ohjeen (engl. tooltip) näkyviin pitämällä osoitinta elementin päällä.	Uudelleennimetään elementti tai muutetaan sen ulkoasua.
Käyttäjät yrittävät saada linkin ohjeen näkyviin tai palaavat linkkiä painettuaan pian takaisin edelliseen näkymään.	Uudelleennimetään linkki.
Käyttäjät joutuvat odottamaan sisällön latautumista ilman palautetta 10 sekuntia tai pidempään.	Näytetään sisällön latautumisen aikana prosessointisivu.
Syötteen kenttä on yleisimpiin käyttötapauksiin nähden joko liian kapea tai leveä.	Muutetaan elementin kokoa.
Palvelin hylkää suuren osan lähetetyistä lomakkeista.	Parannetaan lomakkeen tarkistusta selainpäässä.
Käyttäjät yrittävät usein klikata elementtiä, johon ei liity mitään toiminnallisuutta.	Lisätään elementtiin toiminnallisuus tai muutetaan sen ulkoasua.

Taulukko 1: Käytettävyysongelmia ja ratkaisumalleja niihin

4 Työaikasovelluksen käyttöliittymäsuunnittelu

Tässä luvussa käydään läpi nykyisen työaikasovelluksen periaatteet, käyttöliittymän toiminta ja havaitut ongelmakohdat. Uudelle käyttöliittymälle asetetaan vaatimukset ja kartoitetaan käyttötapaukset. Lopuksi käyttöliittymästä laaditaan luonnoskuvat, joiden pohjalta prototyyppi on mahdollista toteuttaa.

4.1 Sovelluksen käsitteet

Työaikasovelluksen kannalta oleellisia käsitteitä on selitetty taulukossa 2. Näiden lisäksi on syytä mainita myös käsite työpäivä, jonka määrittely on hieman epäselvä. Mikäli työntekijä on esimerkiksi yötyöläinen, jonka työaikakirjaus alkaa maanantaiillalla kello 22.00 ja päättyy tiistaina aamulla 6.00, on tulkinnanvaraista, kumman päivän puolelle tehdyt tunnit kuuluvat. Käyttöliittymä ei ota kantaa tähän ongelmaan, sillä se saa palvelimelta datan valmiiksi päiville jaoteltuna.

Termi	**Selitys**
työntekijä	Organisaation työntekijä, joka tekee sisään- ja ulosleimauksia, yleensä mobiilisovelluksella.
esimies	Työntekijän erikoistapaus, jolla on enemmän oikeuksia. Henkilö voi olla yhden tai useamman osaston esimies.
osasto	Organisaation sisäinen yksikkö. Jokainen työntekijä kuuluu yhteen osastoon. Osasto voi olla toisen osaston alaosasto, jolloin muun muassa esimiesoikeudet periytyvät yläosastolta.
työaikakirjaus	Tapahtuma, joka sisältää aina sisäänleimauksen ja mahdollisesti ulosleimauksen. Molempiin leimauksiin liittyy aikaleiman lisäksi mahdollinen sijaintitieto koordinaattipisteinä ja osoitteena. Kirjaus voi olla tyypiltään normaali työkirjaus tai saldovapaa. Työaikakirjaus, jossa ei ole ulosleimausta, on keskeneräinen.
saldo	Minuuttimäärä, joka kasvaa, kun työntekijä kirjaa tunteja yli työpäivän määrätyn keston. Vastaavasti saldomäärä vähenee, jos työntekijä tekee määrättyä vähemmän tunteja. Työntekijällä on erikseen päiväkohtainen saldo sekä kokonaissaldo. Mikäli saldo alittaa asetetun alarajan, siitä näytetään varoitus.

Taulukko 2: Työaikasovelluksen oleelliset käsitteet

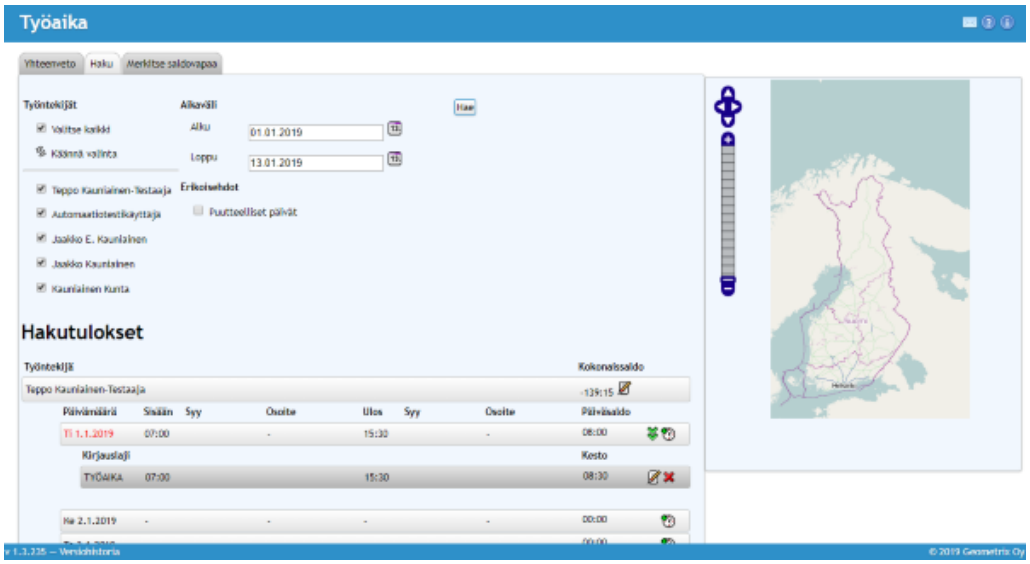
4.2 Nykyisen käyttöliittymän rakenne ja ongelmat

Sovellus on jaettu näkymiin ”Yhteenveto”, ”’Haku” ja ”Merkitse saldovapaa”, joista ensimmäinen on oletuksena aktiivisena. Yhteenvetonäkymän tarkoituksena on, että esimies näkee nopealla vilkaisulla kaikkien osastojensa työntekijöiden kuluvan päivän sisään- ja ulosleimaustiedot, tuntisaldon ja visuaalisen indikaattorin siitä, onko työntekijä tällä hetkellä töissä vai ei. Kokonaistuntisaldoa voi muokata, ja sen ohessa näytetään huutomerkkisymboli, mikäli saldomäärä alittaa sallitun rajan. Tiedot esitetään listakomponentissa, jonka oikealla puolella näytetään kartta. Molempien elementtien rinnakkainen näkyminen vaatii Chrome-selaimella noin 1 350 pikselin leveyden – muutoin karttaelementti tipahtaa listan alle. Käyttöliittymä ei ole responsiivinen eli se ei mukaudu näytön koon mukaan. Sovelluksen käyttö on näin ollen hankalaa kapealla näytöllä tai mobiililaitteella, jolloin kaikki sisältö joko ei mahdu näkyviin tai näkyy niin pienessä koossa, että teksti ei ole luettavaa.

Listakomponentissa ei ole järjestys- tai suodatusmahdollisuuksia, mistä johtuen oikean työntekijän löytäminen vaikeutuu huomattavasti, kun rivien määrä kasvaa. Rivielementteihin on lisätty varjostus luomaan kolmiulotteisuuden vaikutelma, mikä on tyypillisesti toimintopainikkeissa hyödynnettävä tehokeino [Joh14] ja saattaa hämätä käyttäjää luulemaan, että rivin klikkaaminen aiheuttaisi jotain. Ristiriitaisuutta lisää se, että hakunäkymässä on vastaavan näköisiä rivielementtejä, jotka aukeavat klikattaessa. Tästä aiheutuva mahdollinen käytettävyysongelma kuvattiin taulukossa 1.

Kuvassa 2 esitetty hakunäkymä mahdollistaa työntekijöiden työaikakirjausten hakemisen korkeintaan kuukauden ajalta. Lomakkeelta voi valita haettavat työntekijät ja haun aikavälin sekä asettaa ehdon, että vain puutteelliset kirjaukset – esimerkiksi sisäänleimaukset ilman vastaavia ulosleimauksia – haetaan. Kuukauden maksimiaikaraja haussa on oletettavasti asetettu siksi, ettei käyttäjä voisi vahingossa tehdä liian raskasta hakua. Tämä ei kuitenkaan ole järkevä rajaus, sillä esimerkiksi kymmenen työntekijän kuukauden kirjaukset on todennäköisesti suurempi tietomäärä kuin yhden työntekijät kirjaukset vaikkapa kahdelta kuukaudelta.

Hakulomakkeen valintaruutujen (engl. checkbox) aktivointi onnistuu vain valintaruutua itseään klikkaamalla, ei siis esimerkiksi klikkaamalla työntekijän nimeä. Tämä voi hidastaa hakuehtojen asettamista, sillä työntekijöitä valittaessa on osuttava vain reilun kymmenen pikselin levyiseen ja korkuiseen elementtiin. Lomakkeella on oletusarvoisesti valittuna kaikki työntekijät, mikä hankaloittaa tietyn työntekijän rivien hakemista, koska käyttäjän pitää ensin tyhjentää kaikki valintaruudut valit-



Kuva 2: Kuvakaappaus nykyisen käyttöliittymän hakunäkymästä

semalla ”Käännä valinta” ja sitten klikattava valituksi halutun työntekijän valintaruutu.

Hakulomakkeen korkeus riippuu työntekijälistan pituudesta, ja koska hakutulokset näytetään vasta lomakkeen alapuolella, joutuu käyttäjä vierittämään sivua alaspäin nähdäkseen haun tulokset, vaikka listassa olisi vain kymmenen työntekijää. Myös indikaattori käynnissä olevasta hausta jää herkästi piiloon, sillä se näkyy työntekijälistan alapuolella. Lisäksi indikaattori on pelkkä staattinen teksti eikä näin ollen ole mielenkiintoinen katsoa tai anna mielikuvaa siitä, että jokin prosessi on käynnissä.

Yhteenvetonäkymän listan tavoin myöskään hakutuloslistaa ei voi suodattaa tai järjestellä, mikä tekee sen selaamisesta hankalaa. Lista sisältää rivielementtejä kolmella tasolla, ja oletuksena näkyvissä ovat vain haettujen työntekijöiden nimet ja kokonaissaldot. Työntekijärivin voi klikata auki, jolloin sen alle ilmestyy rivi jokaiselle haetulle päivälle. Mikäli työntekijä on tehnyt kyseisenä päivänä työaikakirjauksia, näytetään niistä kootut tiedot. Päivärivin saa edelleen auki ”Näytä kirjaustapahtumat” -painikkeesta, jolloin näkyviin tulevat päivän työaikakirjaukset omille riveilleen eriteltynä. Eri tasojen rivielementit näyttävät samalta, mutta ne toimivat keskenään eri tavalla; siinä missä työntekijärivin saa klikattua auki mistä tahansa kohtaa elementtiä, päivärivin saa avattua vain erillisestä painikkeesta. Tämä on ristiriitaista ja voi hämätä käyttäjää.

Lomakkeet työaikakirjausten ja saldon muokkaamiseen aukeavat modaalisiin dialogi-

ikkunoihin, mikä tarkoittaa sitä, että muun sovelluksen käyttö estyy, kunnes käyttäjä antaa ikkunalle jonkin syötteen. Ikkunassa ei lue, kenen työntekijän kirjausta tai saldoa ollaan muokkaamassa. Tästä seuraa se, että käyttäjä voi klikata vahingossa väärän työntekijän riviä ja tallentaa tiedot huomaamatta virhettä. Virhettä ei välttämättä havaitse tallentamisen jälkeenkään, sillä päivitykset näkyvät vasta, kun rivit hakee uudelleen.

Muokkauslomakkeiden rivien välissä ei ole tyhjää tilaa, ja syötekenttien selitteet on tasattu vasemmalle, mikä vaikeuttaa syötettävien tietojen hahmottamista. Lisäksi toisiinsa liittyvät tiedot, kuten sisäänkirjausaika ja -osoite, eivät sijaitse lomakkeella lähellä toisiaan. Lomake ei myöskään anna välitöntä palautetta virheellisesti syötetystä kellonajasta tai puuttuvasta pakollisesta tiedosta, vaan virheilmoitus tulee vasta, kun käyttäjä yrittää tallentaa lomakkeen.

Dialogien painikerivillä ”Tallenna”-painike sijaitsee ”Sulje”-painikkeen oikealla puolella, mikä poikkeaa muista tuoteperheen sovelluksista. Painikkeet ovat tekstejä lukuun ottamatta identtiset, joten oikean painikkeen löytääkseen käyttäjän on ensin luettava ainakin toisen painikkeen nimi. Suurempi visuaalinen eroavuus auttaisi löytämään halutun painikkeen nopeammin. ”Hyväksy”- ja ”Peruuta”-painikkeiden järjestys kannattaa päättää sen mukaan, käyttääkö enemmistö käyttäjistä sovellusta Windows- vai MacOS-järjestelmällä [Nie08]. Windowsin käyttäjät ovat tottuneet vasemmalla olevaan ”Hyväksy”-painikkeeseen, MacOSin käyttäjät taas päinvastaiseen. Lisäksi useammin käytetystä painikkeesta kannattaa tehdä oletusarvoinen niin, että Enterin painaminen lomakkeella aktivoi sen, ellei kyseessä ole erityisen riskialtis toiminto.

”Merkitse saldovapaa” -välilehti sisältää yksinkertaisen lomakkeen saldovapaan merkitsemiseen tietylle työntekijälle ja päivälle. Saldovapaan kirjaaminen on mahdollista myös työaikakirjauksen syöttölomakkeen kautta. Tämä on kuitenkin hankalampi vaihtoehto, sillä uuden työaikakirjauksen lisääminen työntekijälle onnistuu vain hakunäkymän kautta. Tämä tarkoittaa sitä, että jos puuttuvan kirjauksen haluaa lisätä esimerkiksi kuluvalle päivälle, on käyttäjän siirryttävä yhteenvetosivulta hakuosioon, haettava työntekijän rivit, vieritettävä näkymä kuluvan päivän rivin kohdalle ja klikattava ”Uusi merkintä” -painiketta.

Kun käyttäjä siirtyy edestakaisin sovelluksen välilehtien välillä, kukin näkymä alustuu aina uudelleen. Tästä johtuen esimerkiksi hakulomakkeen valinnat tyhjentyvät, mikäli käyttäjä käy välillä toisessa näkymässä.

Karttaelementti alustuu aina – myös välilehdeltä toiselle siirryttäessä – mittakaa-

vaan, jossa näkyy koko Suomi. Tällainen kartta ei ole informatiivinen organisaatioille, jotka toimivat vain tietyllä alueella, eivät koko maassa. Karttanäkymää voi vapaasti liikuttaa raahaamalla, ja sen mittakaavaa voi suurentaa tai pienentää hiiren rullalla tai karttakäyttöliittymän liukukytkimellä. Käytännössä karttaa kuitenkin tarvitaan vain työaikakirjausten pistemäisten sijaintitietojen esittämiseen. Sisään- ja ulosleimausten sijaintitiedot saadaan näkyviin klikkaamalla karttamerkin näköistä ikonia työaikakirjauksen rivillä.

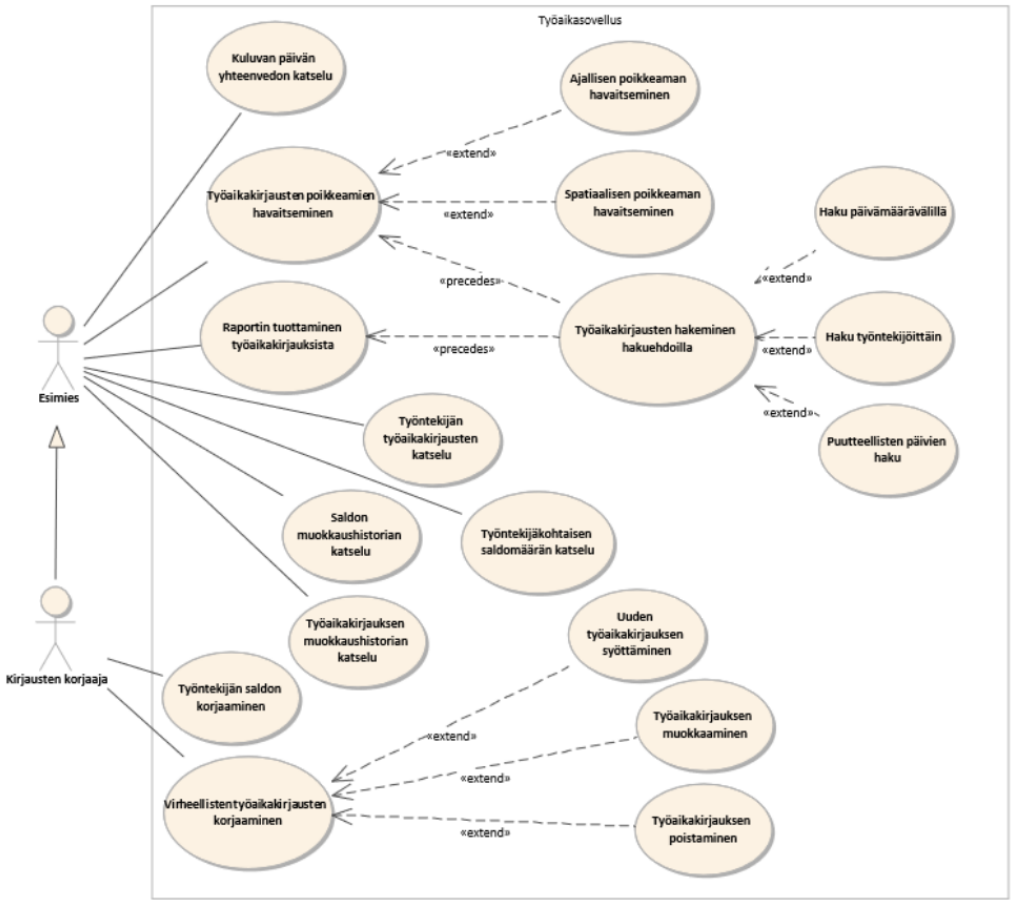
Nykyisen käyttöliittymän ongelmiin ja uuden käyttöliittymän tarjoamiin ratkaisuihin palataan myöhemmin arviointiluvussa.

4.3 Vaatimusmäärittely

Projektin ensimmäinen suunnittelupalaveri pidettiin 11. heinäkuuta 2018. Osallisina olivat tuoteomistaja sekä joukko sovelluskehittäjiä. Uuden käyttöliittymän vaatimukseksi asetettiin, että sen on tuettava 5 000 käyttäjän työaikakirjaustietojen selailua. Käyttöliittymän on toimittava työpöytälaitteen lisäksi myös tabletilla ja tarjottava ainakin jotain toiminnallisuutta myös älypuhelimella. Selaimista käyttöliittymän on tuettava viittä suosituinta, jotka olivat suunnittelupalaverin aikaan Google Chrome, Internet Explorer, Mozilla Firefox, Microsoft Edge sekä Safari [bro18]. Internet Explorerista tuetaan vain versiota 11.

Tuoteomistajan mukaan asiakkaan kannalta oleellista on pystyä helposti tunnistamaan varsinkin systemaattiset poikkeamat työaikakirjauksissa. Poikkeamilla tarkoitetaan joko ajallista poikkeamaa, eli kirjautumista sisään tai ulos liian aikaisin tai myöhään, tai spatiaalista poikkeamaa, jossa työntekijän kirjaukset tapahtuvat kaukana todellisesta työmaasta. Asiakas kuitenkin toivoo, ettei työntekijän sijaintitietoa seurattaisi enempää kuin on välttämätöntä, joten paikkatiedon näyttämiseksi ehdotettiin heat map -tyylistä ratkaisua, joka korostaa kirjausten tiheyttä yksittäisten sijaintipisteiden sijaan.

Aloituspalaverin pohjalta laadittiin kaavio sovelluksen käyttötapauksista. Toimijoita ovat normaali käyttäjä sekä esimieskäyttäjä, joka on erikoistapaus käyttäjästä. Ajatuksena oli, että uusi sovellus olisi käytettävissä esimiesten lisäksi myös työntekijöillä niin, että he näkisivät sieltä omat työaikatietonsa. Mikäli käyttäjällä on esimiesoikeudet, olisi käyttöliittymä laajempi ja sisältäisi kaikkien esimiehen osastojen työntekijöiden tietojen katselun sekä hallinnoinnin. 27.8.2018 pidetyssä palaverissa päätettiin, että ei-esimieskäyttö rajataan tutkielman ulkopuolelle, sillä se monimut-



Kuva 3: Työaikasovelluksen käyttötapaukset

kaistaa käyttöliittymän kehitystä ja työntekijöiden näkymä halutaan mahdollisesti toteuttaa kokonaan omana sovelluksenaan myöhemmin. Kuvassa 3 on lopullinen versio käyttötapauskaaviosta toteutettuna Enterprise Architect -suunnittelutyökalulla.

11.9.2018 pidetyn asiakaspalaverin jälkeisessä sähköpostikirjeenvaihdossa asiakas ilmaisi myös tarpeen pystyä rajaamaan sovelluksen käyttöoikeuksia niin, että kaikki käyttäjät eivät pääsisi muokkaamaan saldo- tai kirjaustietoja. Nykyisessä sovelluksessa ei ole erillistä käyttöoikeuksien rajaamista, vaan käyttäjällä on aina täydet muokkausoikeudet kaikkiin näkemiinsä kirjauksiin ja saldotietoihin. Uusi käyttöoikeusjako on esitetty kuvan 3 käyttötapauskaaviossa erottamalla ”Kirjausten korjaaja” -toimija esimiehen erikoistapaukseksi. Vaikka käyttöoikeuksien rajaamista ei ole tarkoitus toteuttaa vielä prototyyppiin, tiettyjen toimintojen piilottamisen tarve otetaan kuitenkin huomioon suunnittelussa. Lisäksi asiakas esitti toiveen, että esimiehiä koskevat rivit saisi korostettua yhteenvetonäkymässä, ja että saldon muu-

toksesta jäisi loki.

Suunnittelupalaverien pohjalta voidaan vastata alaluvussa 3.6 esitettyihin vaatimusmäärittelyn keskeisiin kysymyksiin. Prototyypin tarkoitus on tarjota nykyistä parempi käyttöliittymä työntekijöiden työaikatietojen katseluun, korjaamiseen ja raportointiin. Tämä tarkoittaa ratkaisuehdotuksien esittämistä alaluvussa 4.2 kirjattuihin nykyisen käyttöliittymän ongelmakohtiin sekä kuvan 3 käyttötapauksiin. Prototyypin tulee olla interaktiivinen ja mahdollistaa esimerkiksi kirjauksen päivittämisen demoaminen riittävällä tasolla, mutta sen ei tarvitse näyttää oikeaa tietoa tai toimia täysin niin kuin lopullinen sovellus toimisi. Käyttöliittymän on toimittava ensisijaisesti työpöytälaitteella, toissijaisesti tabletilla viidellä tällä hetkellä suosituimmalla selaimella. Käyttöliittymäsuunnittelu toteutetaan työpöytälähtöisesti.

Kehitys- ja ylläpitotyön kannalta prototyypin lähdekoodin on oltava riittävän ymmärrettävä myös sellaiselle, joka ei ole aiemmin käyttänyt työhön valittua JavaScript-teknologiaa. Tämä mahdollistetaan sekä selkeällä ja luettavalla koodilla että tarvittaessa ylimääräisellä dokumentaatiolla. Koodin selkeys todennetaan staattisella analyysityökalulla. Koodin on myös oltava automaattisesti testattavissa, jotta pystytään vähentämään aikaa vievän manuaalisen testauksen tarvetta ja virheiden todennäköisyyttä. Testien kattavuuden on oltava todistetusti 100 %, mikä tarkoittaa sitä, että testeissä on käytävä läpi kaikki selaimessa ajettavan koodin funktiot ja haarat. Mikäli kattavuus jää alle tämän arvon, on poikkeukset pystyttävä perustelemaan. Testikattavuus koskee vain itse kirjoitettua koodia, eli siinä ei huomioida liitännäiskirjastoja.

4.4 Käyttöliittymäsuunnitelma

Uudesta käyttöliittymästä laadittiin luonnoskuvat käyttötapauskaavion sekä nykyisen käyttöliittymän ja siitä havaittujen puutteiden pohjalta Pencil Project -työkalulla. Luonnoskuvat on esitetty liitteessä 1. Värit, ikonit, tekstit ja elementtien tarkka asettelu eivät vielä vastaa lopullista. Esimerkiksi toimintopainikkeet on yksinkertaistettu niin, että ”Poista”-painiketta kuvastaa laatikko, jossa on ”P”-kirjain, ja vastaavasti muokkauspainiketta symboloi ”M”-kirjain. Kuvissa ylhäällä näkyvä musta palkki on ”paikanpitäjä” (engl. placeholder) sovellusportaalin navigaatiolle, joka ei kuulu prototyyppiin mutta joka tulee sisältymään lopulliseen käyttöliittymään.

Yleiseltä rakenteeltaan käyttöliittymä mukailee nykyisen sovelluksen ulkoasua, sillä perusrakenteessa ei ole havaittu ongelmia ja käyttäjät ovat oletettavasti tottuneet

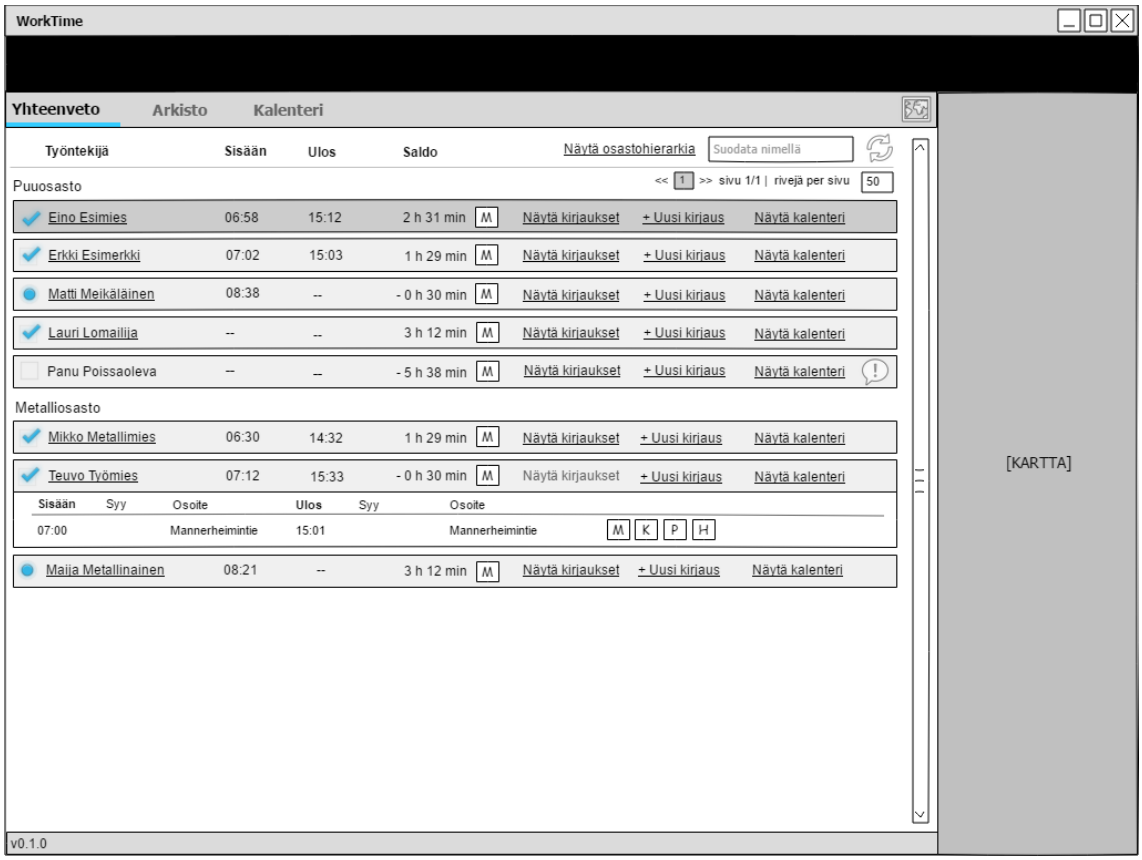
siihen. Sovelluksen nykyisille käyttäjille on todennäköisesti kehittynyt mentaalinen malli siitä, miltä tämän sovelluksen kuuluu näyttää, joten liian suuret muutokset käyttöliittymässä voisivat hankaloittaa heidän osaltaan käyttöä. Käyttöliittymä on jaettu kahteen lohkoon – varsinaiseen sisältöön ja karttaan – niin, että lohkojen välistä suhdetta on mahdollista muuttaa hiirellä vetämällä. Oletuksena karttalohko vie leveyssuunnassa noin yhden viidesosan käyttöliittymästä. Kartan saa tarvittaessa kokonaan piiloon tai takaisin näkyviin ”Piilota/näytä kartta” -painikkeella, joka on vasemman lohkon oikeassa ylänurkassa. Tieto kartan näkyvyydestä ja lohkojen suhteesta tallennetaan selaimen evästeisiin niin, että mikäli käyttäjä ei halua nähdä karttakuvaa ollenkaan, tarvitsee se piilottaa vain ensimmäisellä käyttökerralla.

Sivun ylälaidassa sijaitseva navigaatio ja vasemmalle painottuva pääsisältö myös mukailevat alaluvussa 2.3 mainittua F-kaavaa, joka tukee tyypillistä tapaa selata verkkosivuja. Toisaalta karttanäkymän voisi tulkita kuvaksi, jolloin sen sijoittaminen tekstimuotoisen pääsisällön vasemmalle puolelle mukailisi puolestaan sitä, miten ihmisen näkökentän puoliskot erikoistuvat kuvien ja tekstin tulkintaan.

Vasen lohko on jaettu kolmeen vaihtoehtoiseen näkymään, joiden välillä voi navigoida vasemman yläreunan välilehtipainikkeilla. Kuten nykyisessäkin käyttöliittymässä, ensimmäinen ja oletusarvoinen näkymä on ”Yhteenveto”, joka näyttää listana työntekijöiden kuluvan päivän työaikatiedot. Yhteenvetonäkymän luonnos on esitetty kuvassa 4. Käyttöliittymän skaalautuvuutta suurelle työntekijämäärälle on parannettu lisäämällä listaan sivutus, suodatus ja ja mahdollisuus järjestää rivit eri sarakkeiden mukaan. Näkymässä näytetään kerrallaan esimerkiksi 20 työntekijää, ja lisää rivejä nähdäkseen käyttäjän on siirryttävä seuraavalle sivulle. Käyttäjä voi itse muuttaa yhdellä sivulla näytettävien rivien lukumäärää.

Listan rivejä voi suodattaa kirjoittamalla vapaavalintaisen merkkijonon ”Suodata nimellä” -tekstikenttään, jolloin työntekijöistä jäävät näkyviin vain ne, joiden nimestä löytyy annettu merkkijono. Tämä mahdollistaa tietyn työntekijän hakemisen pitkästäkin listasta nopeasti, mikäli nimi on tiedossa. Kolmantena skaalautuvuutta tukevana ominaisuutena on listan järjestäminen sarakkeittain esimerkiksi työntekijän nimen mukaan aakkosjärjestykseen tai saldomäärän mukaan niin, että ensimmäisenä listassa näkyvät ne työntekijät, joilla on eniten negatiivista saldoa.

Nykyisten ”Sisällä”- ja ”Ei sisällä” -indikaattorisymbolien lisäksi uudessa yhteenvetonäkymässä näytetään ”Kirjautunut ulos” -ikoni niille työntekijöille, joilla on kuluvalle päivälle sekä sisään- että ulosleimaus. Näin saadaan erotettua työpäivän päättäneet työntekijät niistä, jotka eivät ole ollenkaan kirjautuneet sisään. Työntekijäriveille



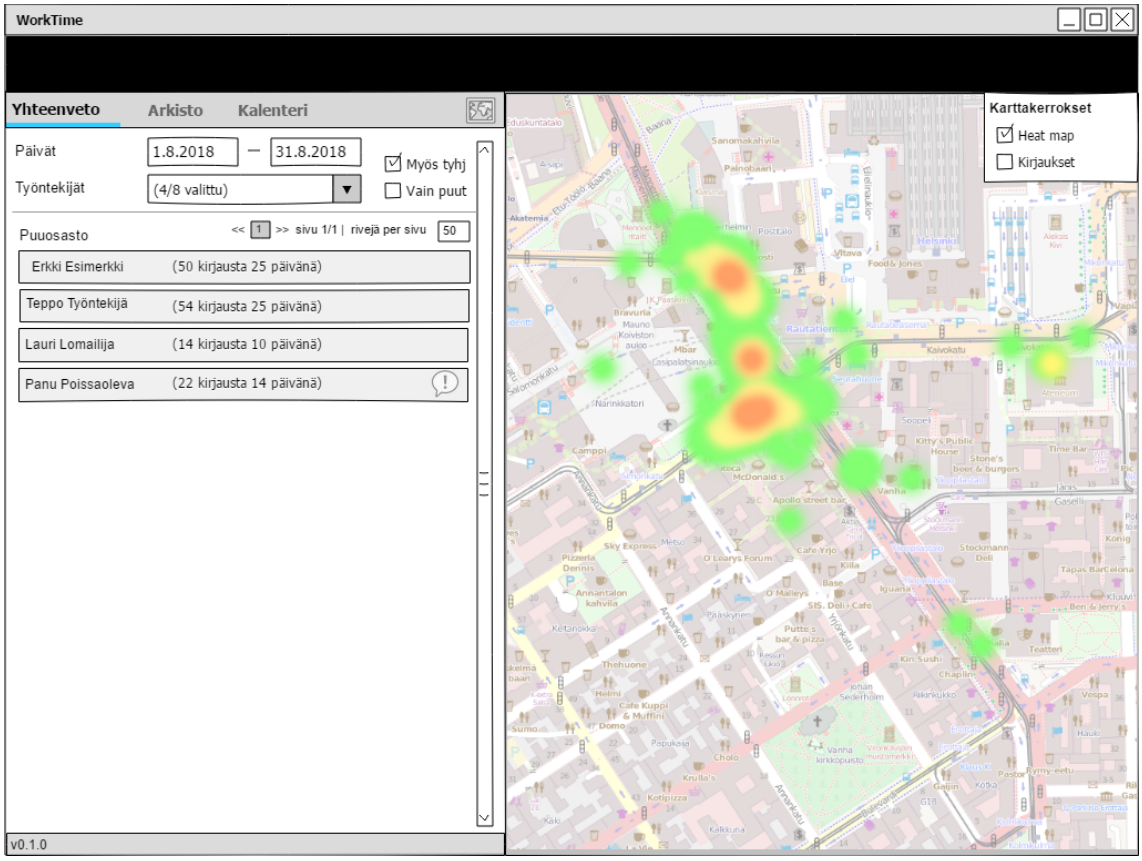
Kuva 4: Käyttöliittymäluonnos yhteenvetonäkymästä

on lisätty pikalinkit uuden työaikakirjauksen lisäämiseen sekä kalenterinäkymään. Tämä nopeuttaa uuden kirjauksen tekemistä, koska ei ole tarvetta siirtyä hakunäkymään ja tehdä erillistä hakua.

Työkirjausten ja saldon muokkaamisen näkymät ovat yksi käyttöliittymän haasteista. Niiden toteuttaminen dialogi-ikkunoina nykyisen sovelluksen tapaan olisi yksinkertaisempi ratkaisu kuin se, että lomakkeet aukeaisivat osaksi pääkäyttöliittymää. Modaalisen ikkunan haittapuolia on muun muassa se, että kontekstin yllättävä muuttuminen voi saada käyttäjän unohtamaan, mitä oli tekemässä [Fes17]. Ongelmaa pahentaa se, että dialogi voi peittää oleellista sisältöä muusta käyttöliittymästä, johon palatakseen käyttäjän on ensin suljettava dialogi. Modaaliset ikkunat eivät myöskään toimi hyvin pienillä näytöillä etenkään vierityspalkkien kanssa [Mat14]. Dialogi-ikkunoita kehotetaan käyttämään pääasiassa vain kriittisen informaation tai lomakkeen esittämiseen [Fes17, Dou18].

Vanha sovellus näyttää ainoastaan käyttäjän osaston työntekijöiden tiedot. Koska

esimiehen alaisuudessa voi kuitenkin olla useita osastoja ja niiden alaosastoja, näytetään uudessa käyttöliittymässä kaikkien näihin kuuluvien työntekijöiden tiedot osastoittain ryhmiteltynä. ”Näytä osastohierarkia” -painike tuo näkyviin käyttöoikeuksien mukaisen osastolistan esitettynä hierarkkisessa listassa. Osastojen näkyvyys on mahdollista kytkeä päälle tai pois päältä valintaruuduilla, mikäli käyttäjä haluaa tarkastella vain tietyn osaston tietoja.



Kuva 5: Käyttöliittymäluonnos hakunäkymästä ja avatusta kartasta

Yhteenvetosivun tavoin myös ”Haku”-näkymä on toteutettu suurelta osin vanhan käyttöliittymän pohjalta. Hakulomakkeelta valitaan haettavat työntekijät ja päivämäärärajaus, minkä lisäksi käyttäjä voi valita haettavaksi pelkästään puutteelliset työaikakirjaukset. Työntekijöiden listaus, joka tekee vanhasta hakukäyttöliittymästä vaikeakäyttöisen, on toteutettu tilan säästämiseksi avattavana valintalistana, josta haettavat työntekijät voi valita. Työntekijät on listattu osastoittain niin, että valinta on helppo asettaa päälle tai pois päältä kaikille tietyn osaston työntekijöille. Uutena hakuehtona on ”Myös tyhjät päivät” -valintaruutu, jonka kytkeminen pois päältä mahdollistaa pelkästään työaikaleimauksia sisältävien päivien hakemisen. Nykyinen

haku noutaa aina kaikki päivät aikarajauksen sisältä, mikä hankaloittaa oleellisten rivien löytämistä. Tyhjät päivärivit ovat käyttäjälle oleellisia lähinnä silloin, kun aikomuksena on lisätä puuttuva työaikakirjaus tietylle päivälle.

Haetut työntekijät esitetään listana, joka on yhteenvetonäkymän tavoin jaoteltu osastojen mukaan. Oletuksena työntekijästä näytetään nimi ja hakuehtoja vastaavien työaikaleimausten lukumäärä. Työntekijärivin klikkaaminen avaa listan työaikakirjauksista. Toisin kuin nykyisessä käyttöliittymässä, työpäiviä ei esitetä omina riveinään. Sen sijaan päivämäärät näytetään listan vasemmassa laidassa, ja saman päivän sisällä tehdyt kirjaukset erotetaan muista kirjauksista erotinviivojen avulla.

Hakutuloslistan lisäksi haetut työaikaleimaukset näkyvät myös kartalla. Käyttäjä voi valita kahdesta vaihtoehtoisesta karttakerroksesta, joista toinen näyttää leimausten pistemäiset sijainnit ja toinen esittää sijaintidatan lämpökarttana (engl. heat map), jossa paljon leimauksia sisältävät alueet näkyvät punaisena, vähiten edustetut alueet sinisenä ja muut muina spektrin väreinä. Lämpökartta antaa yleiskuvan siitä, missä työaikaleimauksia tehdään, ja auttaa havaitsemaan poikkeamat, kuten jatkuvasti työmaan ulkopuolella tehdyt leimaukset.

Myös hakutuloslistasta käsin on mahdollista lisätä työntekijöille puuttuvia työaikakirjauksia tai korjata virheellisiä kirjauksia. Yhteenvetonäkymässä on mahdollista käsitellä vain kuluvan päivän kirjauksia, kun taas hakunäkymässä korjauksia voidaan tehdä vanhoihinkin merkintöihin. Puuttuvan kirjauksen lisääminen tietylle päivälle onnistuu hakemalla kirjauksia aikaväliltä, johon kohdepäivä sisältyy.

”Kalenteri”-välilehti on kokonaan uusi näkymä, joka on käytännössä vain erilainen tapa visualisoida työaikakirjauksia tietyltä aikaväliltä. Kalenteri näyttää käyttäjän valitseman työntekijän työaikakirjaukset, joita voi selata kuukausittain eteen ja taakse. Puutteelliset päivät näkyvät taustavärillä korostettuna. Kirjausten muokkaaminen ja lisääminen on mahdollista kuten muissakin näkymissä. Koska kalenterinäkymä ei sisällä mitään tietoa tai tärkeää toiminnallisuutta, mitä ei olisi saatavilla muualla sovelluksessa, sen toteuttaminen voidaan jättää projektissa viimeiseksi tai tarvittaessa rajata kokonaan sen ulkopuolelle.

Uudessa käyttöliittymässä näytetään työkirjausten ja saldon historiatiedot, joita asiakas on toivonut. Tämä ominaisuus jäi kuitenkin pois käyttöliittymäluonnoksista.

4.5 Palvelinrajapinta

Tiedonsiirto selaimen ja palvelimen välillä toteutetaan noudattaen REST-arkkitehtuuria (Representational State Transfer). REST-verkkopalvelussa määritellään joukko resursseja, joista kullakin on yksikäsitteinen osoite, joka vastaanottaa määrätynlaisia HTTP-pyyntöjä [PZL08]. REST on käytössä myös muissa yrityksen uusimmissa sovelluksissa, ja se on selainpään kannalta vaivaton ottaa käyttöön, mistä johtuen se sopii hyvin prototyypin rakentamiseen.

Tässä alaluvussa laaditaan REST-rajapinnan kuvaus, jonka pohjalta pyynnöt ja vastausten käsittely toteutetaan prototyyppiin. Rajapinta on kokonaisuudessaan esitetty taulukossa 3. Ensimmäisessä sarakkeessa on kuvattu REST-resurssin suhteellinen osoite, esimerkiksi ”/appdata”, ja toisessa sarakkeessa vaadittu HTTP-metodi. ”GET”-tyyppiset pyynnöt ovat tiedon lukemiseen, kun taas ”POST” lähettää tietoa palvelimelle ja ”DELETE” välittää tiedon poistopyynnön. Palvelin lähettää ja vastaanottaa tietoa JSON-muodossa.

URL	**Metodi**	**Kuvaus**
/appdata	GET	Palauttaa sovelluksen versionumeron
/configuration	GET	Palauttaa mm. työkirjausten tyyppikoodiston sekä karttaan liittyvät määritykset
/hierarchy	GET	Palauttaa osastohierarkian
/workers	GET	Palauttaa työntekijät ja kuluvan päivän yhteenvetotiedot
/archive	GET	Palauttaa työntekijät ja hakuparametreja vastaavat työaikakirjaukset
/report	GET	Luo parametrejä vastaavista kirjauksista Excel-raportin
/records	GET	Palauttaa hakuparametrejä vastaavat kirjaukset
/records	POST	Lisää uuden kirjauksen
/records/:id	GET	Palauttaa kirjauksen
/records/:id	POST	Päivittää kirjausta
/records/:id	DELETE	Poistaa kirjauksen
/balance/:wid	GET	Palauttaa kokonaissaldon
/balance/:wid	POST	Päivittää kokonaissaldoa

Taulukko 3: REST-rajapinnan kuvaus

Ensimmäinen rajapintakutsu hakee palvelimelta vain sovelluksen versionumeron. Käyttöliittymä renderöidään vasta, jos tämä pyyntö onnistuu ja palauttaa vastauksen; sitä ennen sovelluksesta näytetään vain käynnistyskuva (engl. splash screen).

Mikäli pyyntö epäonnistuu, käyttäjä ohjataan virhe- tai sisäänkirjautumissivulle eikä enempää rajapintapyyntöjä suoriteta. Tämä varmistaa sen, että käyttöliittymää ei edes yritetä näyttää, mikäli yhteys palvelimeen ei ole kunnossa. Virhetilanne saattaisi ilmetä esimerkiksi silloin, kun käyttäjä on todellisuudessa kirjautunut ulos sovelluksesta, mutta selain hakee sovelluksen HTML- ja JavaScript-koodin muistista ja saa näin renderöityä ainakin osittaisen käyttöliittymän, jota käyttäjän ei kuuluisi nähdä ollenkaan.

Ensimmäisen pyynnön onnistuttua rajapinnasta kysellään ne tiedot, joita tarvitaan käyttöliittymän alustuksessa. Tässä vaiheessa käyttäjän selaimessa on jo näkyvissä osittainen käyttöliittymä, josta puuttuvat vain ne osat, joiden tietoja ei ole vielä saatu palvelimelta. Esimerkiksi hakulomakkeelle siirtyminen ja hakuehtojen täyttäminen on mahdollista ilman, että palvelimelta on vielä saatu mitään muuta kuin yllä mainittu ensimmäinen vastaus. Tämän ansiosta käyttäjä pystyy aloittaamaan sovelluksen käytön ja tehtävien suorittamisen mahdollisimman pian. Käyttäjä myös saa nopeammin palautetta kuin siinä tapauksessa, että käyttöliittymässä näytettäisiin pelkkä latausindikaattori siihen asti, että kaikki näkymät on saatu alustettua.

Työaikatietojen esittämistä varten tarvitaan osastohierarkia eli tieto kaikista osastoista ja työntekijöistä, joihin käyttäjällä on katseluoikeus. Koska järjestelmän tietomallissa organisaatio sisältää aina vain yhden juuritason osaston, jonka alla kaikki muut alaosastot sijaitsevat, tieto osastohierarkiasta voidaan palauttaa ylimmän tason osastoa kuvaavana JSON-oliona. Olio sisältää osaston tietokantatunnisteen ja nimen sekä listan kaikista osaston työntekijöistä ja alaosastoista, jotka ovat samanlaisia osasto-olioita kuin vanhempansa. Osastohierarkia haetaan vain kerran käyttöliittymän alustuksen yhteydessä, koska sen ei oleteta muuttuvan usein. Ei siis ole todennäköistä päätyä tilanteeseen, jossa jokin osasto poistetaan ja työaikasovelluksen käyttäjät näkevät hierarkian, joka ei ole ajan tasalla.

Seuraavassa on esimerkki yhden osaston kuvauksesta JSON-oliona. Merkintä "..." tarkoittaa alaosastojen listaa, joka on jätetty tästä yksinkertaistetusta esityksestä pois.

```
{
  "id": 200,
  "name": "Esimerkkiosasto",
  "childDepartments": [
    {
      "id": 201,
      "name": "Alaosasto",
      "childDepartments": [ ... ]
     }
  ]
}
```

Osaston JSON-kuvaus ei sisällä tietoa sen työntekijöistä. Prototyypin toteutusvaiheessa kävi ilmi, että osasto- ja työntekijätiedot kannattaa pitää käyttöliittymässä erillään, sillä työntekijöiden esittäminen osastoille alisteisina osoittautui toteutusvaiheessa hankalaksi. Tämä käydään tarkemmin läpi alaluvussa 5.6. Prototyypin käyttöliittymän kannalta oleellista on vain, että työntekijäolio tietää työntekijän osaston tietokantatunnisteen, jotta työntekijälistaa voidaan suodattaa osaston perusteella.

Yhteenvetonäkymää varten tarvitaan sekä tieto kaikkien osastojen kaikista työntekijöistä että näiden työntekijöiden kuluvan päivän kirjaus- ja saldotiedot. ”workers”-rajapinta palauttaa täydellisen työntekijälistan, jossa kuhunkin JSON-olioon sisältyy työntekijän tunnistetietojen lisäksi yhteenvetotiedot. Samaa listaa käytetään uudelleen arkistonäkymän hakulomakkeen työntekijävalinnassa, vaikka kyseinen valintalista ei tarvitse yhteenvetotietoja mihinkään. Tällä vältetään se, että täydellinen työntekijälista pitäisi kysellä palvelimelta useamman kerran, mikä kasvattaisi sovelluksen vasteaikoja.

Kun käyttäjä klikkaa yhteenvetonäkymän työntekijärivin auki, prototyyppi noutaa ”records”-rajapinnasta tarkemmat tiedot kuluvan päivän yksittäisistä työaikakirjauksista. Tämä ratkaisun tavoitteena on nopeuttaa ensimmäistä työntekijäkyselyä, mutta se saattaa osoittautua turhaksi, sillä palvelinpuoli joutuu joka tapauksessa hakemaan tietokannasta yksittäiset työaikakirjaukset selvittääkseen päivän ensimmäisen ja viimeisen leimauksen tiedot. Prototyyppiin kuitenkin jätetään tämä toimintamalli, sillä se helpottaa käyttöliittymään suorituskyvyn testaamiseen vaadittavan testidatan generointia, josta puhutaan myöhemmin alaluvussa 5.4.

Arkiston hakutoimintoa varten tarvitaan mahdollisuus kysellä työaikakirjauksia niin, että pyynnössä annetaan kyselyparametreinä muun muassa haettavat työntekijät sekä aikaväli. Koska haetut kirjaukset halutaan esittää alisteisina työntekijöille, ”records”-rajapinta ei sovellu tähän. Arkistokyselyjä varten on oma ”archive”-raja-

pintansa, joka palauttaa listan työntekijäolioita, joiden lapsina on lista hakuehtoihin täsmäävistä työaikakirjauksista. Koska ”records”-rajapinta palauttaa pelkkiä kirjauksia, pitäisi kirjausten yhdistäminen työntekijäriveihin toteuttaa selainpäässä. Tämä todennäköisesti muodostuisi selainpuolen suorituskykyongelmaksi suurella tietomäärällä. Näin ollen on järkevämpää jättää kirjausten ja työntekijöiden kytkeminen toisiinsa palvelinpään vastuulle. Rajapintamääritys tosin olisi selkeämpi, jos olisi vain yksi rajapinta työntekijätietojen hakemiseen ja yksi rajapinta kirjausten hakemiseen.

Seuraavassa on kuvattu JSON-olio, joka sisältää yksittäisen työaikakirjauksen tiedot.

```
{
  "id": 12345,
  "type": 1,
  "signInTime": "2018-11-28 07:58.12:345",
  "signInAddress": "Mannerheimintie 1, Helsinki",
  "signInCoords": [0, 0].
  "signOutTime": "2018-11-28 16:02:45.678",
  "signOutAddress": "Mannerheimintie 1, Helsinki",
  "signOutCoords": [0, 0],
  "version": 0,
  "history": [ ... ]
}
```

Työaikakirjauksen JSON-olioon sisältyvät kirjauksen ja sen tyypin tietokantatunnisteet, sisään- ja ulosleimausten tiedot, kirjaustietueen versionumero sekä historiatiedot listana, joka on edellä esitetty viitteellisesti muodossa ”[...]”. Sisään- ja ulosleimaustiedot voisivat olla myös erillisiä, rakenteeltaan samanlaisia JSON-olioita, mutta toteutusvaiheessa kävi ilmi, että kirjaustietojen esittäminen käyttöliittymässä on helpompaa, mikäli kaikki tiedot ovat oliossa samalla tasolla.

Kirjauksen muutoshistoria on mukana oliossa listamuotoisena tietona. Lista kannattanee kuitenkin jättää tyhjäksi silloin, kun kirjaus haetaan näkymään, jossa muutoshistoriaa ei tarvitse näyttää, sillä historian hakeminen todennäköisesti lisää palvelinpään kuormaa huomattavasti suuria kirjausmääriä käsitellessä. Prototyypissä riittää, että muutoshistoria palautetaan vain yksittäisen kirjauksen tietoja hakiessa eli kun kysely tehdään kirjauksen tietokantatunnisteella.

Yhteenvetonäkymän työntekijäolio sisältää tiedon vain kokonaissaldon minuuttimäärästä, mutta saldoon liittyy lisäksi metatietoja sen edellisestä päivityksestä. Täydellinen saldo-olio haetaan siksi erillisellä kyselypyynnöllä ”balance”-rajapinnasta. Saldon päivityspyynnössä ei kuitenkaan lähetetä samaa oliota, vaan tällöin selain lä-

hettää ainoastaan saldon muutoksen lukuarvona, muutoksen lisätiedot sekä versionumeron. Selain ei siis koskaan pyydä asettamaan uutta kokonaissaldoa eksplisiittisesti, vaan antaa ainoastaan tiedon lisäyksestä tai vähennyksestä. Ratkaisu perustuu oletukseen, että loppukäyttäjälle on helpompaa syöttää saldon muutos ja antaa järjestelmän hoitaa uuden kokonaissaldomäärän laskeminen. Saldon päivitysolio on kuvattu alla olevassa JSON-esimerkissä.

```
{
  "balanceChange": -30,
  "notes": "Korjataan vahinkokirjaus",
  "version": 1
}
```

Versionumero oliossa mahdollistaa sen, että palvelin voi verrata selaimen lähettämän olion versiota tietokannasta löytyvän vastaavan tietueen versionumeroon ja estää tallennuksen sekä varoittaa käyttäjää konfliktista, mikäli riviä on ehditty päivittää toisaalta käsin sen jälkeen, kun se on haettu selaimeen. Tällä saavutetaan se hyöty käytettävyydessä, että mikäli kaksi esimiestä yrittää vahingossa tehdä samaa kirjauksen tai saldon korjausta samaan aikaan, jälkimmäinen päivittäjä saa virheilmoituksen ja tiedon siitä, että kyseistä tietuetta on jo päivitetty toisaalta käsin. Mikäli versionumeroita ei verrattaisi, saattaisi edellä mainitussa tilanteessa tapahtua niin, että kun kaksi käyttäjää haluaa kirjata saman puolen tunnin saldovähennyksen, vähentyykin saldosta yhteensä tunti.

4.6 Käyttöliittymän komponentit

Käyttöliittymäluonnoksista voidaan tunnistaa sovelluksen ulkoasun komponentit. Komponenteista pyritään tekemään mahdollisimman monikäyttöisiä eli sellaisia, että uuden samanlaisen komponentin voi pienellä vaivalla lisätä käyttöliittymään joko sellaisenaan tai mukailtuna. Työntekijälista on esimerkki elementistä, joka esiintyy käyttöliittymässä useassa eri paikassa hieman erilaisena. Todennäköisesti on kehitys- ja ylläpitotyön suhteen kannattavaa, että yhteenvetonäkymä, hakutuloslista ja hakulomakkeen valintalista käyttävät samaa komponenttimääritystä, sillä kaikkiin näihin listoihin halutaan muun muassa sivutus ja suodatus. Mikäli esimerkiksi sivutuksen logiikkaa halutaan korjata tai päivittää, riittää, että muutoksen tekee yhteiseen komponenttimääritykseen eikä kolmeen erilliseen komponenttiin.

Käyttöliittymän komponenttihierarkian ylimmällä tasolla on koko sovelluksen kattava juurikomponentti, jonka alikomponentteja kaikki muut käyttöliittymän elementit

ovat. Juurikomponentin alla on komponentit kartalle ja vasemmalle lohkolle, joka edelleen jakautuu ylä- ja alapalkkeihin sekä välilehtikohtaiseen näkymään. Se, miten hienojakoinen komponenttihierarkiasta lopulta tulee, määräytyy vasta toteutusvaiheessa. Mikäli yhden komponentin määrittävä tiedosto käy liian pitkäksi, on se mahdollisesti syytä jakaa pienempiin osiin, jotta koodin luettavuus on parempaa.

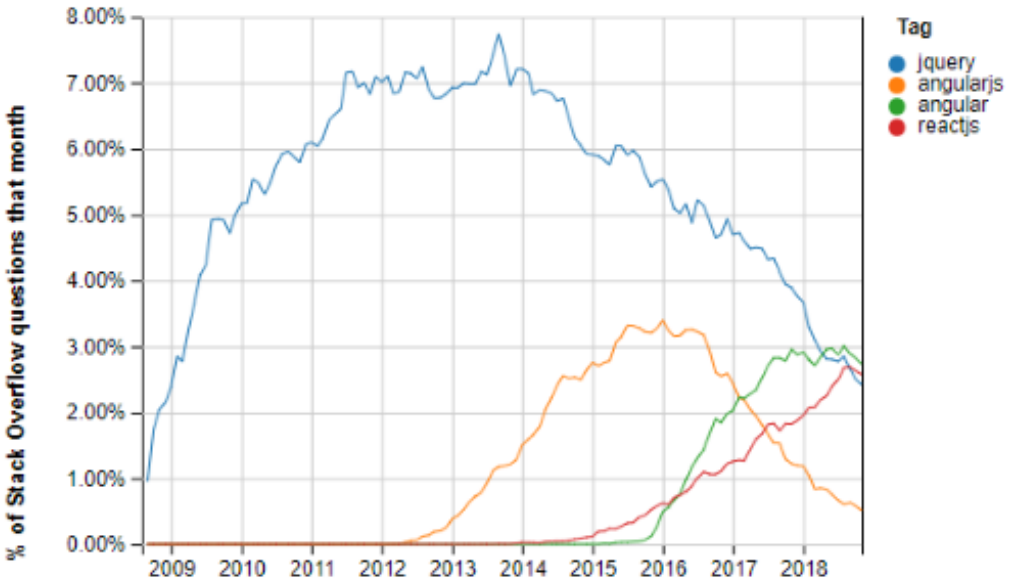
Esimerkiksi käyttöliittymän painikkeet ovat yksinkertaisia elementtejä eivätkä vaadi erillistä komponenttimäärittelyä, vaan pelkkä yhteinen tyylimäärittely riittää. Kaikille niille HTML-elementeille, joiden halutaan näyttävän täysin tai osittain samalta, määritellään yhteinen CSS-luokka. Jos halutaan esimerkiksi muuttaa kaikkien painikkeiden väriä, tarvitsee tämän ansiosta muutos tehdä vain yhdelle tyylitiedoston riville.

5 Työaikasovelluksen käyttöliittymäprototyyppi

Tässä luvussa valitaan kehistyössä käytettävät teknologiat ja työkalut ja laaditaan toimiva käyttöliittymäprototyyppi uudelle työaikasovellukselle. Prototyypistä tulee itsenäinen selainpuolen sovellus, joka tekee rajapintakutsuja testirajapintaan palvelimen sijaan. Prototyyppi pyritään toteuttamaan niin, että sitä voidaan pienin muutoksin hyödyntää myös asiakkaalle julkaistavassa tuotantoversiossa.

5.1 JavaScript-teknologiat

JavaScript on monimutkainen ja hankalasti ymmärrettävä ohjelmointikieli, mistä johtuen selainpuolen kehitystä on pyritty helpottamaan erilaisilla sovelluskehyksillä ja kirjastoilla, jotka tarjoavat valmiita rakenteita ja suuntaviivoja sovelluksen rakentamiseen [fus18]. Sovelluskehittäjä Ian Allen [All18] kertoo blogikirjoituksessaan, että JavaScript-sovelluskehysten ja -kirjastojen elinkaari on usein hyvin lyhyt, sillä uusia teknologioita kehitetään jatkuvasti.

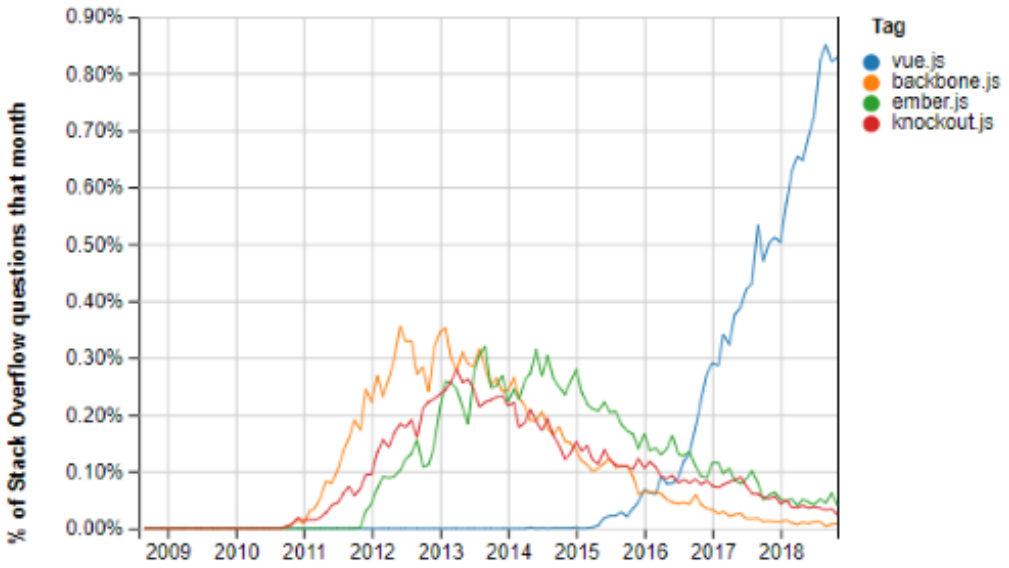


Kuva 6: JS-teknologioiden suosion kehitys Stack Overflow -sivustolla [sta19]

Stack Overflow Trends -työkalulla [sta19] on mahdollista piirtää kuvaajia, jotka havainnollistavat eri teknologioiden esiintymistä sivustolla esitetyissä kysymyksissä vuodesta 2008 nykyhetkeen. Kuvassa 6 on vertailtu neljän suositun JavaScript-teknologian tilannetta keväällä 2019. JQuery-kirjasto on näistä teknologioista van-

hin ja saavuttanut aikanaan selkeästi suurimman suosion, mutta sen esiintyvyys kysymyksissä lähti tasaiseen laskuun samaan aikaan, kun AngularJS-sovelluskehys ilmestyi. AngularJS:n suosio saavutti huippunsa 2016, jonka jälkeen myös sen suosio alkoi laskea uudemman Angular-teknologian ja Reactin ilmaannuttua. Alkuvuonna 2019 on nähtävissä pientä laskua sekä Angularin että Reactin suosiossa, mutta tämä voi olla ohimenevä heilahdus.

Kuvassa 7 on puolestaan vertailtu muutamien vähemmän suosittujen JavaScript-teknologioiden esiintyvyyttä. Kuvaajasta näkee, että suosion kasvu on alussa ollut nopeaa mutta lähtenyt parin vuoden päästä laskuun. Vertailluista teknologioista uusin on Vue.js, jonka suosio on kasvanut jyrkästi viime vuosina ja näkyy olevan edelleen kasvussa keväällä 2019. Vanhempien teknologioiden elinkaarista on kuitenkin pääteltävissä, että myös tämän sovelluskehyksen suosio saattaa ennemmin tai myöhemmin kääntyä laskuun.



Kuva 7: JS-teknologioiden suosion kehitys Stack Overflow -sivustolla [sta19]

Kuvaajia tulkitessa on huomioitava se, että Stack Overflow -kysymysten määrä on vain yksi tapa mitata teknologian suosiota. Laskeva kysymysten määrä voi kertoa esimerkiksi siitä, että teknologiasta on jo saatavilla niin paljon tietoa, että uusien kysymysten esittäminen ei ole tarpeellista.

Nykyinen työaikasovelluksen käyttöliittymä käyttää JSP-teknologiaa, mikä tarkoittaa sitä, että HTML-sivut muodostetaan jo palvelimella. Käyttöliittymän interaktiivisuus on toteutettu JQuery-kirjastolla. Toimeksiantajayrityksen uudemmat so-

vellukset hyödyntävät selainpuolella AngularJS-sovelluskehystä. Muissa kehitysprojekteissa on kuitenkin havaittu AngularJS:n rajoitteet etenkin pitkien, dynaamisten listojen esittämisessä, sillä listojen alustaminen ja päivittäminen on raskasta. Tästä on syntynyt tarve kartoittaa vaihtoehtoisia JavaScript-kirjastoja työaikasovelluksen prototyypin kehitystä varten. Teknologiavertailuun on valittu aikaisemman kokemuksen perusteella AngularJS, tämän seuraaja Angular, suosittu React-kirjasto sekä uudempi sovelluskehys Vue.js. Kaikki teknologiat tukevat vaadittujen selainten uusimpia versioita joko sellaisenaan tai ylimääräisen kirjaston avulla [ang18b, ang18a, rea18, vue19].

Googlen ylläpitämä AngularJS on teknologiavertailun sovelluskehyksistä vanhin. Kehittäjien mukaan se on suunniteltu ajatellen erityisesti tiedon luontiin, lukemiseen, päivittämiseen ja poistoon perustuvia sovelluksia [ang18b]. Kehys perustuu ajatukseen, että deklaratiivinen koodi toimii imperatiivista koodia paremmin käyttöliittymien rakentamisessa. Käytännössä tämä tarkoittaa sitä, että yksittäisiä HTML-elementtejä ei muokata eksplisiittisesti, vaan käyttöliittymän muutokset kuvataan deklaratiivisesti niin, että elementit muuttuvat sovelluksen tilan muuttuessa. Myös muut vertailtavat teknologiat ovat deklaratiivisia. JQuery taas on esimerkki kirjastosta, jossa dokumenttioliomallia muokataan suoraan.

AngularJS pyrkii vähentämään vaadittavan koodin määrää hyödyntämällä muun muassa kaksisuuntaista datan sitomista (engl. ”two-way data binding”) [TB14]. Muutokset tietomalleissa heijastuvat automaattisesti käyttöliittymään, ja vastaavasti käyttäjän vuorovaikutus käyttöliittymän kanssa – esimerkiksi tekstikenttään kirjoittaminen – päivittää tietomallia. Näin ollen ei ole tarvetta kirjoittaa erillistä logiikkaa tiedon synkronoimiselle toisin kuin esimerkiksi JQueryssä. Toinen kehykselle tyypillinen ominaisuus ovat direktiivit, jotka ovat uudelleenkäytettäviä HTML-elementtien laajennoksia. Esimerkiksi listaelementin toteuttaminen direktiivinä tarkoittaa sitä, että käyttöliittymään voidaan helposti lisätä monta eri listaa, jotka käyttävät samaa lähdekoodia sellaisenaan tai parametrisoituna. Nämä ominaisuudet vähentävät koodin määrää ja helpottavat näin sen ylläpidettävyyttä sekä vähentävät virheiden todennäköisyyttä, mikä parantaa käytettävyyttä.

AngularJS tukee myös yksisuuntaista datan sitomista, jossa vain tietomalli päivittää käyttöliittymää, sekä kertaluontoista datan sitomista, jossa tietomallin arvot päivitetään käyttöliittymään vain kerran, tyypillisesti käyttöliittymän alustuksessa. Kertaluontoinen sitominen on suorituskyvyn kannalta tehokkaampi ratkaisu kuin kaksi- ja yksisuuntainen sitominen, sillä se ei vaadi tietomallin ja käyttöliittymän

muutosten seuraamista. Kertaluontoinen sitominen on käytännöllnen ratkaisu niissä tapauksissa, joissa tietoa tarvitsee päivittää harvoin tai ei ollenkaan [Gre16].

Angular – johon luetaan tässä sen versiot 2, 4 ja 5 – on AngularJS:n seuraaja, joka on kokonaan uudelleen kirjoitettu ja enimmäkseen epäyhteensopiva edeltäjänsä kanssa. Tästä johtuen AngularJS:n tuntemuksesta ei ole suurta etua Angularin oppimisessa. Angular hyödyntää komponenttihierarkiaa, mikä on samankaltainen ratkaisu kuin Facebookin kehittämässä React-kirjastossa. [ang18b]

React käyttää virtuaalista dokumenttioliomallia, joka on perinteistä DOMia kevyempi ratkaisu. Suurin osa sovelluskehyksistä vuorovaikuttaa suoraan selaimen generoiman dokumenttioliomallin kanssa, mikä tarkoittaa DOMin puurakenteen läpikäymistä jokaisen muutoksen yhteydessä ja vaikuttaa negatiivisesti suorituskykyyn ja edelleen käytettävyyteen erityisesti suurta tietomäärää päivittäessä. React puolestaan päivittää suoraan vain virtuaalista dokumenttioliomallia, joka on muistissa oleva varsinaisen DOMin abstraktio. Päivityksen jälkeen React selvittää muuttuneet elementit vertaamalla virtuaalista ja varsinaista dokumenttioliomallia keskenään, jolloin se voi päivittää ainoastaan tietyt elementit koko puurakenteen sijaan. [Agg18]

Itsenäisen sovelluskehittäjän Evan Youn luoma Vue.js perustuu Angularin ja Reactin tavoin komponenttihierarkiaan. AngularJS on toiminut Vue.js:n inspiraationa, mikä näkyy syntaksin samankaltaisuudessa. Reactin tavoin Vue.js mahdollistaa datan kulkemisen komponenttien välillä vain yhteen suuntaan, mutta mahdollistaa kuitenkin kaksisuuntaisen datan sitomisen komponentin sisällä, esimerkiksi lomakkeiden syötekentissä. Myös Vue.js hyödyntää virtuaalista dokumenttioliomallia. [vue19]

Stefan Krause [Kra17] tarjoaa blogissaan työkalun eri JavaScript-teknologioiden suorituskyvyn vertailuun. Työkalu luo taulukon satunnaisesti generoidulla datalla ja laskee, kauanko kullakin sovelluskehyksellä menee aikaa erilaisissa taulukkoon liittyvissä operaatioissa. Taulukkoon 4 on kirjattu työkalun tuottamat tulokset AngularJS:n versiolle 1.6.3, Angularin versiolle 4.1.2, Reactin versiolle 15.5.4 ja Vue.js:n versiolle 2.3.3. Operaatioiden kestot ovat millisekunteina ja sulkuihin on kirjattu keskihajonta. Sivun alustuksella tarkoitetaan sitä aikaa, mikä menee JavaScript-koodin lataamiseen ja jäsentelyyn sekä sivun renderöimiseen. Neuhaus [Neu17] kuitenkin huomauttaa, että suorituskykytestiin on suhtauduttava kriittisesti, sillä sovelluskehys on voitu optimoida saamaan hyviä tuloksia tällaisessa testissä.

Mlynarskin ja Nurzynskan kokeellisessa tutkimuksessa [MN17] vertaillaan Reactin, AngularJS:n ja Angularin suorituskykyä sivun renderöinnin suhteen sekä työpöytä-

operaatio	**AngularJS**	**Angular**	**React**	**Vue.js**
1000 rivin luominen	251,8(8,0)	193,1(7,9)	212,2(14,2)	166,7(8,6)
1000 rivin päivittäminen	273,8(16,7)	197,4(5,4)	206,7(7,3)	168,5(5,0)
joka 10. rivin päivittäminen	12,5(2,0)	13,0(4,5)	18,0(1,6)	17,3(2,9)
klikatun rivin korostaminen	8,1(3,6)	3,4(2,3)	8,7(2,9)	9,3(1,7)
kahden rivin vaihtaminen keskenään	14,7(2,4)	13,4(1,0)	17,1(1,3)	18,3(1,5)
rivin poistaminen	47,4(2,4)	46,1(3,2)	52,4(1,7)	52,6(2,7)
10 000 rivin luominen	3 108,7(2 162,2)	1 946(41,8)	1 931,7(35,6)	1 587,5(33,9)
1000 rivin lisääminen 10 000 rivin taulukkoon	454,8(42,1)	324,6(10,1)	366,4(10,9)	399,5(11,0)
10 000 rivin tyhjentäminen	817,6(37,2)	379,9(11,3)	410,9(9,8)	254,5(5,0)
sivun alustus	118,1(5,1)	83,4(2,6)	93,8(6,9)	56,6(2,5)

Taulukko 4: JS-teknologioiden suorituskyky eri taulukko-operaatioissa

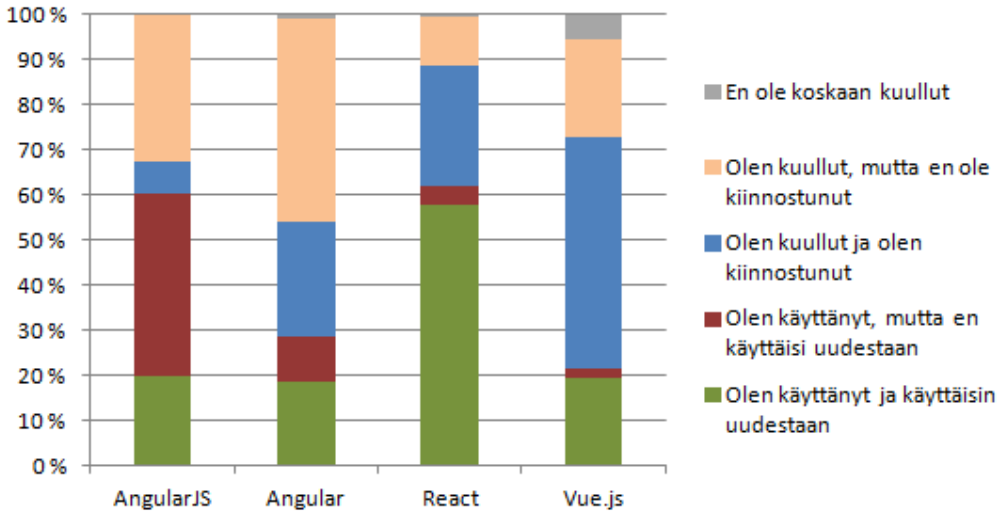
että mobiililaitteella. Teknologiat pisteytettiin sen mukaan, miten ne suoriutuvat lista- ja lomake-elementtien esittämisessä sekä näkymästä toiseen siirtymisessä. Tutkimuksen yhteenvedossa AngularJS:lle on annettu 105 pistettä, Angularille 98 ja Reactille 84,5. Korkeimmat pisteet työpöytäversiolla testatessa sai Angular, kun taas AngularJS suoriutui kokonaisuudessaan parhaiten mobiiliversion esittämisessä.

Web-sovelluksen suorituskykyä voidaan parantaa sillä, että sivun renderöinti tapahtuu selainpuolen sijaan etukäteen palvelinpuolella. Kaikki vertaillut teknologiat AngularJS:ää lukuun ottamatta tarjoavat kirjastot tällaisille ratkaisuille [Neu17].

Neuhausin [Neu17] mukaan Angular on teknologioista haastavin oppia kattavasta dokumentaatiosta huolimatta. Usein AngularJS-sovellusten parissa työskennellyt Hermant Rai [Rai17] perustelee siirtymistä Vue.js:ään sillä, että sovellusten kasvava tietomäärä heikensi AngularJS:n suorituskykyä liikaa. Rain mukaan suorituskykyongelmat katosivat Vue.js:n käyttöönoton myötä, ja se on ollut kaikista hänen kokeilemistaan JavaScript-sovelluskehyksistä helpoin oppia. Myös Petrosyan [Pet18] arvioi Vue.js:n helpoimmaksi ja Angularin vaikeimmaksi oppia vertaillessaan Vue.js:ää, Reactia ja Angularia. Petrosyanin mukaan React on yksinkertainen oppia, mutta se vaatii useita muita kirjastoja tuekseen, kun taas Vue.js tarjoaa valmiin kokonaisuuden. Malhotra [Mal17] kuvailee Vue.js:ää Angularia kevyemmäksi, yksinkertaisemmaksi ja joustavammaksi. Kaimer ja Brune [KB18] kehuvat Vue.js:ää sekä Angularia että Reactia kevyemmäksi. FusionChartsin vertaileva artikkeli [fus18] kritisoi Angu-

laria jyrkästä oppimiskäyrästä ja suosittelee Vue.js:ää erityisesti pienille kehitystiimeille, mutta huomauttaa, että uutena sovelluskehyksenä Vue.js:lle ei ole tarjolla yhtä paljon yhteisön tukea kuin vanhemmille teknologioille.

The State of Javascript 2017 -sivustolle [sta17] on koottu tulokset JavaScript-teknologioiden käyttöä koskevasta kyselystä, johon on vastannut yli 28 000 kehittäjää. Kuvan 8 kaaviossa on esitetty vastausprosentit tässä tutkimuksessa käsiteltäville teknologioille. Stack Overflow -sivuston vuoden 2017 kyselyyn [sur17] vastasi yli 64 000 kehittäjää, joista 44,3 prosenttia vastasi käyttävänsä AngularJS:ää tai Angularia ja 19,5 prosenttia Reactia. JetBrainsin loppuvuoden 2016 ja alkuvuoden 2017 välillä toteutetun kyselyn [BGR17] mukaan sovelluskehittäjistä Reactia käyttää 49 prosenttia, AngularJS:ää 44 prosenttia, Angularia 22 prosenttia ja Vue.js:ää 20 prosenttia, mutta kyselyyn vastasi vain vähän yli 5 000 kehittäjää.



Kuva 8: JavaScript-teknologiakyselyn tuloksia [sta17]

5.2 JavaScript-teknologian valinta

Pano, Graziotin ja Abrahamsson selvittivät haastattelututkimuksessaan [PGA18], mitkä tekijät ovat kehittäjille oleellisia JavaScript-teknologian valitsemisessa. Näihin lukeutuvat muun muassa teknologian suorituskyky, opittavuus, soveltuvuus, laajennettavuus, kehitysyhteisön aktiivisuus ja sosiaalinen vaikutus. Vaadittavan toiminnallisuuden on oltava toteutettavissa niin, että se ei vaadi liikaa suorituskyvyn resursseja – kuten muistia ja siirtonopeutta – tai tuota liian montaa riviä koo-

dia. Opittavuus kuvastaa sitä vaivannäön määrää, mikä kehittäjältä vaaditaan uuden teknologian sisäistämiseen. Käyttöönoton haastavuus riippuu sekä teknologian kompleksisuudesta ja sen dokumentaation kattavuudesta että kehittäjän osaamisesta. Muutamat haastateltavat kertoivat valitsevansa teknologian sen mukaan, kuinka paljon aikaa projektille on annettu.

Soveltuvuus kuvastaa sitä, missä määrin tuote sopii asetettujen vaatimusten täyttämiseen. Kehittäjät suosivat sellaista sovelluskehystä, joka tarjoaa kaikki tarpeet kattavan ratkaisun. Tämä säästää aikaa ja vaivaa suhteessa siihen, että toiminnallisuuden toteuttamiseksi jouduttaisiin etsimään ja hyödyntämään useita eri kirjastoja, joiden on oltava keskenään yhteensopivia. Laajennettavuus puolestaan viittaa siihen, että sovelluskehyksen pitäisi olla tarpeeksi joustava, jotta siihen voidaan tarvittaessa liittää ulkoisia kirjastoja. [PGA18]

Teknologiasta vastaavan kehitysyhteisön koko ja aktiivisuus ovat tärkeitä valintatekijöitä, sillä niiden perusteella on mahdollista tunnistaa tuotteet, joiden elinkaaren voi odottaa olevan pitkä ja joita tullaan kehittämään jatkuvasti. Mitä suurempi yhteisö, sitä nopeammin virheilmoituksiin ja kehitystoiveisiin vastataan. Tästä johtuen vanhoja sovelluskehyksiä suositaan usein suhteessa uusiin. Sosiaalisella vaikutuksella tarkoitetaan muiden ihmisten – esimerkiksi kollegoiden ja kilpailijoiden – teknologiavalintojen ja mielipiteiden huomiointia. Sovelluskehys arvioidaan luotettavaksi, jos se on käytössä muissa saman alan yrityksissä. [PGA18]

Taulukossa 4 vertailtiin sovelluskehysten suorityskykyjä Krausen [Kra17] työkalulla mitattuna. Tämän perusteella mikään vertailluista teknologioista ei ole yksiselitteisesti muita tehokkaampi, mutta Vue.js suoriutuu parhaiten suuren rivimäärän luonnissa ja tyhjentämisessä sekä sivun alustuksessa. AngularJS ja Angular ovat muita tehokkaampia taulukon osittaisessa päivittämisessä, valitun rivin korostamisessa sekä kahden rivin vaihtamisessa keskenään. Angular on selvästi tehokkaampi kuin edeltäjänsä, joka suoriutuu erityisen huonosti rivien luonnissa ja tyhjentämisessä sekä sivun alustuksessa. React-kirjasto ei tarjonnut parasta suorituskykyä missään mitatussa operaatiossa. Yllättävää on, että Mlynarskin ja Nurzynskan kokeessa [MN17] AngularJS sai huomattavasti Reactia paremmat pisteet ja ylsi lähes seuraajansa tasolle. Aggarwal [Agg18] kuitenkin kuvailee Reactin suorituskykyä erittäin hyväksi.

Kuvassa 8 esitettyjen loppuvuoden 2017 kyselytulosten perusteella React on teknologioista selvästi suosituin ja sen kehittäjät ovat myös kaikista tyytyväisimpiä valitsemaansa kirjastoon: yli puolet kyselyyn vastanneista on käyttänyt Reactia ja

voisi käyttää sitä myös jatkossa. Vastaajista lähes yhtä moni on käyttänyt AngularJS:ää, mutta kokemukset siitä ovat huomattavasti negatiivisempia. Tämä mahdollisesti heijastuu siihen, että vastaajista valtaosa ei ole kokeillut Angularia eikä ole kiinnostunut siitä.

Vuoden 2018 kyselytuloksissa ei ole suuria muutoksia suhteessa edelliseen vuoteen, mutta Angularin, Reactin ja Vue.js:n tyytyväisten käyttäjien osuus on kasvanut [sta18]. Tuloksissa ovat mukana myös teknologioiden käyttöosuudet yrityskokojen mukaan. Vue.js on suosituin valinta 1–5 ja 10–20 työntekijän yrityksissä, minkä perusteella se sopisi toimeksiantajayritykselle. 9.2.2019 tarkasteltuna Vue.js on suosituin myös GitHubin tähtiluokituksen eli projektia seuraavien käyttäjien määrän perusteella: satoihin tähtiin pyöristettynä Vue.js:llä on 127 400 tähteä, Reactilla 121 900 tähteä, AngularJS:llä 59 400 tähteä ja Angularilla 45 200 tähteä. Osallistuvia kehittäjiä on ollut eniten AngularJS:llä ja Reactilla, kun taas Vue.js:llä osallisia on ollut selkeästi vähiten, vain 260 verrattuna AngularJS:n 1 600:aan.

	AngularJS	**Angular**	**React**	**Vue.js**
Määritelmä	Sovelluskehys	Sovelluskehys	Kirjasto	Sovelluskehys
Julkaisuvuosi	2010	2016	2013	2014
Pääkehittäjä	Google	Google	Facebook	Evan You
Tähtiä GitHubissa	59 400	45 200	121 900	127 400
Osallistujia GitHubissa	1 600	840	1 280	260
Komponenttipohjainen	Ei	Kyllä	Kyllä	Kyllä
Datan sitominen	2-suuntainen	2-suuntainen	1-suuntainen	2-suuntainen

Taulukko 5: Vertailtavien JS-teknologioiden ominaisuuksia

Vertailtavien teknologien ominaisuudet on kiteytetty taulukkoon 5. Vertailun ja henkilökohtaisen kiinnostuksen perusteella tämän projektin sovelluskehykseksi valitaan Vue.js. AngularJS rajataan pois heikon suorituskyvyn takia ja Angular puolestaan jyrkän oppimiskäyrän vuoksi. React ja Vue.js ovat vertailun perusteella molemmat potentiaalisia vaihtoehtoja, ja valinta näiden välillä tapahtui tämän tutkielman tekijän omien mieltymysten pohjalta. AngularJS:ää muistuttava syntaksi voi olla oppimista helpottava tekijä, sillä toimeksiantajayrityksellä on aikaisempaa kokemusta AngularJS:n käytöstä. Vue.js kuvataan usein helposti opittavaksi, joten se todennäköisesti sopii hyvin prototyypin rakentamiseen nopeahkossa aikataulussa. Teknologian loiva oppimiskäyrä parantaa käyttöliittymän ylläpidettävyyttä, mikä puolestaan edistää käytettävyyttä sitä kautta, että käyttöä sujuvoittavien ominaisuuksien

kehittäminen on helpompaa.

5.3 Muut työkaluvaatimukset

Projektin versionhallinta on yrityksen muiden tuotteiden tapaan GitHub-palvelussa, joka käyttää Git-versionhallintajärjestelmää. GitHub tarjoaa työkalut koodin arviointiin, mikä sekä parantaa koodin laatua että auttaa virheiden löytämistä ennen sovelluspäivitysten julkaisemista asiakkaalle.

JavaScript-kirjastoriippuvuuksien hallinta toteutetaan NPM-paketinhallintasovelluksella (Node Package Manager), joka on käytössä myös muissa yrityksen uudemmissa projekteissa ja jota myös Vue.js:n oma käyttöohje suosittelee [vue19]. NPM on osa Node.js:ää, joka on palvelinpuolen JavaScript-ympäristö. NPM:n perustana on ”package.json”-tiedosto, johon kirjataan muun muassa projektin metatiedot sekä lista kirjastoriippuvuuksista ja näiden versioista.

Monista JavaScript-kirjastoista julkaistaan uusia versioita hyvin tiheään tahtiin. Jotta varsinkin kirjastojen virhekorjaukset saadaan tämän projektin käyttöön mahdollisimman nopeasti, on tärkeää ylläpitää riippuvuuskirjastojen versioita. Tähän tarkoitukseen on olemassa ainakin kirjasto ”npm-check-updates”, jolla saa sekä tarkistettua riippuvuuksien uusimmat versiot että otettua ne käyttöön.

Kehitystyön nopeuttamiseksi ja sujuvoittamiseksi projektiin tarvitaan automaatiotyökalu, joka hoitaa esimerkiksi lähdekooditiedostojen kokoamisen, kääntämisen ja minifioinnin. Yrityksen nykyiset projektit hyödyntävät Grunt-tehtävänsuorittajaa (engl. task runner). Yksi vaihtoehtoinen työkalu tähän on Gulp. 4.1.2019 Gulpilla oli GitHubissa yli 30 000 tähteä ja Gruntilla vajaa 12 000, mikä kertoo edellä mainitun suuremmasta suosiosta.

Tehtäviä suoritetaan sekä kehittäjän toimesta tämän omassa ympäristössä että automaattisesti Jenkins-palvelimella, johon projektille määritellään oma julkaisuputki (engl. release pipeline). Kun kehittäjä vie muutoksensa paikallisesta säilöstään GitHubissa sijaitsevaan säilöön, Jenkins käynnistää putken automaattisesti ja ajaa määrätyt komennot. Putki voi esimerkiksi tehdä sovelluksesta käännöksen, ajaa automaatiotestit, julkaista sovelluksen testausympäristöön ja lopulta viedä julkaisukelpoisen paketin tuotantopalvelimelle. Putken voi määritellä niin, että ajo keskeytyy, mikäli esimerkiksi testit tai koodin laatutarkistus eivät mene läpi. Tämän ansiosta virheelliseksi havaittu versio ei päädy ikinä tuotantoon asti.

NPM:n lisäksi Vue.js:n virallinen käyttöohje [vue19] suosittelee käytettäväksi Web-

pack- tai Browserify-moduulinhallintatyökalua (engl. module bundler). Näiden tarkoituksena on modulaarisen lähdekoodin ja sen riippuvuuksien yhdistäminen yhdeksi, selaimessa suoritettavaksi JavaScript-tiedostoksi [Gim16]. Käyttöliittymän alustus nopeutuu, kun käyttäjän selaimen tarvitsee ladata vain yksi tiedosto usean sijaan.

Käyttöliittymän karttakomponentti toteutetaan Gekko-työkalun avulla. Gekko on yrityksen sisäisessä käytössä oleva JavaScript-kirjasto, joka puolestaan hyödyntää avoimen lähdekoodin OpenLayers-karttakirjastoa. Etuna pelkkään OpenLayersiin verrattuna on se, että Gekko tarjoaa yksinkertaistetun rajapinnan sekä valmiin käyttöliittymän esimerkiksi karttakerrosten näkyvyyden hallintaan.

Prototyypin on tarkoitus toimia kehitys- ja arviointivaiheessa itsenäisenä sovelluksena, joka ei ole riippuvainen palvelinpäästä. Selaimen ja palvelimen välinen tiedonsiirto simuloidaan laatimalla testausrajapinta, joka vastaa kaikkiin selainpuolen tekemiin verkkopyyntöihin. Testirajapintatoteutus paketoidaan osaksi prototyyppiä. Projektiin tarvitaan HTTP-pyynnöt toteuttava JavaScript-kirjasto, joka voidaan tarvittaessa asettaa käyttämään testirajapintaa.

Testaustyökaluista Vue.js:n käyttöohje [vue19] mainitsee Karma-testausympäristön, joka mahdollistaa koodin testaamisen eri selaimilla ja laitteilla, sekä testauskehykset Jestin ja Mochan. 18.1.2019 tarkasteltuna Jestillä on GitHubissa vajaa 23 000 tähteä ja Mochalla vajaa 17 000. Vue.js myös tarjoaa oman yksikkötestauskirjastonsa ”Vue Test Utils” [vue18]. Testien kattavuus on pystyttävä dokumentoimaan, jotta voidaan todentaa toteutuksen laatu testikattavuuden suhteen.

Lähdekoodin virheiden etsimisessä on hyötyä myös koodin staattisesta analysointityökalusta (engl. lint), jolla voidaan havaita mahdollisesti häiriötilanteeseen johtavien virheiden lisäksi myös esimerkiksi käyttämättömiä koodinpätkiä ja muuttujia [Jon02]. Näin ollen työkalun käytöllä voidaan edesauttaa koodin toimivuuden lisäksi sen pysymistä siistinä ja luettavana, mikä edesauttaa ylläpidettävyyttä.

5.4 Projektin perustaminen

Prototyypin varsinainen kehitystyö aloitettiin 2.1.2018 luomalla GitHubiin uusi säilö (engl. repository). Säilö pidetään erillään olemassa olevan työaikasovelluksen säilöstä, jotta selain- ja palvelinpuolta on mahdollista kehittää erikseen. Projektin säilö sisältää ”src”-kansion lähdekoodille sekä ”test”-kansion JavaScript-testeille. Lähdekoodikansiossa on projektin pääasiallinen JavaScrip-tiedosto ”index.js”, tyyliohjeet

kokoava Less-tiedosto ”index.less” sekä ”index.html”, joka on koko käyttöliittymän HTML-pohja. Lisäksi Vue.js-komponentit eli käyttöliittymän eri osaset sijaitsevat ”components”-alikansiossa. Tarkoituksena on, että lähdekoodikansion tiedostoista käännetään ”build”-kansioon varsinainen, selaimessa suoritettava koodi. Käännöskansiota ei sisällytetä versionhallintaan, sillä se luodaan uudestaan jokaisen käännöksen yhteydessä.

Projektin kääntäminen vaatii Vue.js:n komponenttien ja muiden skriptien kokoamisen yhdeksi JavaScript-tiedostoksi, Less-tyylitiedostojen kääntämisen CSS-tyyliohjeeksi ja näiden tiedostojen sisällyttämisen ”index.html”-tiedostoon. Käännöksen automatisointia kokeiltiin aluksi Gulp-tehtävänsuorittajalla, joka on aiemmin yrityksessä käytössä ollutta Gruntia suositumpi ja vaikutti syntaksiltaan yksinkertaisemmalta. Jokaisen tehtävän suorittaminen vaati kuitenkin oman lisäosansa, jolloin ”package.json”-tiedoston riippuvuuslista kasvoi nopeasti hyvin suureksi. JavaScript-koodin kääntäminen onnistui helpoimmin Webpack-työkalulla, ja lopulta kävikin ilmi, että Webpack korvaa Gulpin täysin vaadittavien tehtävien suorittamisessa. Webpack yhdistää samaan tiedostoon sekä JavaScriptin että tyylimäärittelyt, jolloin käyttäjän selaimeen ladataan ”index.html”-tiedoston ja kuvien lisäksi vain tämä koontitiedosto, mikä nopeuttaa sivun latautumista.

HTTP-pyyntöjen toteuttamiseen valittiin Axios-kirjasto. ”Vue-Resource” on aikaisemmin ollut pääasiallinen HTTP-kirjasto Vue.js:lle, mutta kehittäjä Evan You kertoo vuoden 2016 blogikirjoituksessaan [vue16], ettei kyseinen kirjasto enää ole virallinen suositus. Sen sijaan hän ehdottaa Axios-kirjastoa uusille käyttäjille. Mock-rajapinnan toteuttamiseen valittiin kirjasto ”axios-mock-adapter”, koska sen sai helposti toimimaan yhteen Axiosin kanssa.

Jotta prototyypin suorituskyky ja käytettävyys voidaan todentaa vaaditulla 5 000 työntekijällä, täytyy testausrajapinnan palauttaa ainakin tämän verran rivejä yhteenveto- ja arkistonäkymään. Tarvittava määrä riittävän oikean muotoista dataa saatiin generoitua Mockaroo-verkkosivulla.

Lähdekoodin staattiseen analysointiin käytetään ESLint-nimistä JavaScript-lintteriä. ESLintin toiminta perustuu konfiguraatiotiedostoon, johon kootaan kaikki toimeenpantavat säännöt. Sääntölistan pohjaksi valittiin ”eslint-config-standard”-konfiguraation versio 12.0.0, johon lisättiin Vue.js:n virallinen ESLint-lisäosa ”eslint-plugin-vue”. Muutama säännöistä ylikirjoitettiin sillä perusteella, että ne todettiin epäoleellisiksi tai niiden suhteen sovelletaan mieluummin vaihtoehtoista sääntöä. Esimerkiksi koodin sisennyksessä käytettävien välilyöntien määräksi asetettiin oletusar-

vona olleen kahden sijaan neljä.

Testaamisen työkaluiksi valittiin Vue Test Utils ja Jest. Vue.js:n oma testauskirjasto mahdollistaa dokumenttioliomallin testaamisen komponenttitasolla, ja sillä saa simuloitua esimerkiksi painikkeen napsauttamisen. Jest puolestaan suorittaa kirjoitetut testit ja kertoo, mitkä testeistä menivät läpi ja mitkä tuottivat väärän lopputuloksen. Jestin lisäksi kokeiltiin Mochaa, mutta kävi ilmi, että Jest suoriutui samojen testien ajamisesta nopeammin: Mochalla kesti 15 testissä 7–8 sekuntia, kun taas Jest suoriutui tehtävästä keskimäärin 4 sekunnissa. Lisäksi Jestin mukana tulee työkaluja dokumenttioliomallin emulointiin ja testaamiseen sekä testikattavuuden todentamiseen. Mochan käyttö taas vaatii näiden työkalujen hakemisen erikseen.

Kuvassa 9 on esimerkki Jestin tuottamasta taulukosta, jossa on kuvattu kunkin lähdekoodiluokan testikattavuus lauseittain, haaroittain, funktioittain ja riveittäin. Vihreällä värillä on korostettu ne rivit, joissa testikattavuus on hyvä eli vähintään 80 prosenttia. Keltaisilla riveillä kattavuus on heikompi. Oikeimmanpuoleinen sarake näyttää ne luokan rivit, jotka vielä vaativat testaamista. Projektin vaatimusmäärittelyn mukaan testikattavuuden on oltava kokonaisuudessaan 100 %, kun prototyyppi on valmis.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	90.43	100	81.58	90.43	
src	100	100	100	100	
filters.js	100	100	100	100	
src/components	89.41	100	81.08	89.41	
app.vue	85.71	100	66.67	85.71	48,51
header.vue	50	100	50	50	21
main-content.vue	100	100	100	100	
pagination.vue	100	100	100	100	
resizer.vue	100	100	100	100	
summary-row.vue	63.64	100	40	63.64	60,61,63,64
summary.vue	90	100	90.91	90	100,101

Kuva 9: Keskeneräisen projektin testikattavuus Jestin mukaan

Prototyypin JavaScript-lähdekoodi kirjoitetaan hyödyntäen ECMAScript 2015:n mukaista syntaksia. ECMAScript on kielistandardi, jota myös JavaScript noudattaa [ecm15]. Vuoden 2015 versio on standardin toistaiseksi laajamittaisin päivitys, joka tarjoaa ohjelmointia helpottavia ja selkeyttäviä uusia ominaisuuksia. Sekä automaattisten testien suorittaminen että Internet Explorer -selaimen tukeminen kuitenkin vaativat sen, että lähdekoodi käännetään vanhempaan JavaScript-muotoon. Tämä toteutetaan Babel-nimisellä JavaScript-kääntäjällä [bab18].

5.5 Käyttöliittymän perusrakenne

Reactin virallinen dokumentaatio [rea18] esittelee prosessin, jolla yksinkertainen React-sovellus rakennetaan määritellyn JSON-rajapinnan ja käyttöliittymäsuunnittelijan luonnoksen pohjalta. Käyttöliittymäsuunnitelmasta tunnistetaan yksittäiset komponentit sillä periaatteella, että yhden komponentin pitäisi olla vastuussa vain yhdestä asiasta; mikäli komponentin toiminta laajenee,se pitäisi jakaa pienempiin alikomponentteihin. Kun komponentit ja niiden hierarkia on tunnistettu, voidaan sovelluksesta rakentaa ensimmäinen, staattinen versio, jossa ei vielä ole interaktiivisuutta. Dokumentaation mukaan suuremmissa projekteissa on yleensä helpointa aloittaa rakentaminen hierarkiassa alimpana olevista komponenteista.

Toisin kuin Reactin dokumentaatio ehdottaa, tämän prototyypin rakentaminen aloitettiin hierarkian ylätasolta, jotta saatiin käyttöliittymän kokonaisrakenne toteutettua ensin. Kehittäjällä ei ole aikaisempaa kokemusta komponenttihierarkiaan perustuvan käyttöliittymän toteutuksesta, joten perusrakenteesta aloittaminen tuntui luontevammalta ratkaisulta. Perusrakenteen tekeminen alkuun mahdollistaa myös sen, että toimivaa käyttöliittymää on mahdollista esitellä esimerkiksi tuoteomistajalle jo kehityksen aikaisemmassa vaiheessa. Myös komponenttien interaktiivisuutta alettiin toteuttaa saman tien, mikä mahdollisti Vue.js:n ominaisuuksiin tutustumisen heti toteutuksen alkuvaiheessa. Mikäli valittu sovelluskehys ei olisikaan toteuttanut vaadittua toiminnallisuutta riittävän hyvin, olisi toteutuksen alkuvaiheessa ollut mahdollista vaihtaa toiseen työkaluun.

Prototyypin perusrakenne vastaa käyttöliittymäluonnoksissa kuvattua. Käyttöliittymä on jaettu kahteen lohkoon, joista vasemmassa on varsinainen sisältö välilehdille jaettuna ja oikeassa karttakomponentti. Kartan saa tarvittaessa piiloon vasemman lohkon yläpalkin painikkeesta. Lohkojen suhdetta on mahdollista muuttaa tarttumalla hiirellä niiden väliseen reunaan ja vetämällä vaakasuoraan. Kyseessä on yksisivuinen sovellus, jossa päänäkymä ei ikinä muutu, vaan ainoastaan vasemmassa lohkossa näytettävä sisältö päivittyy. Siirtymät välilehtien välillä ovat nopeampia kuin nykyisessä sovelluksessa, sillä koko sivua ei tarvitse renderöidä joka kerta uudestaan. Vasteajat nopeutuvat myös sen takia, että käyttöliittymän rakentamisen hoitaa selain eikä palvelin.

Työntekijälista, osastohierarkia ja muut näkymien vaatimat tiedot haetaan palvelimelta heti käyttöliittymän alustuksen jälkeen. Pyynnöt tehdään asynkronisesti taustalla, jolloin käyttöliittymän muut osiot ovat käytettävissä, vaikka jotain tietoa ei olisikaan vielä saatu palvelimelta. Näkymissä, jotka odottavat edelleen palvelimen

vastausta, näytetään latausindikaattori.

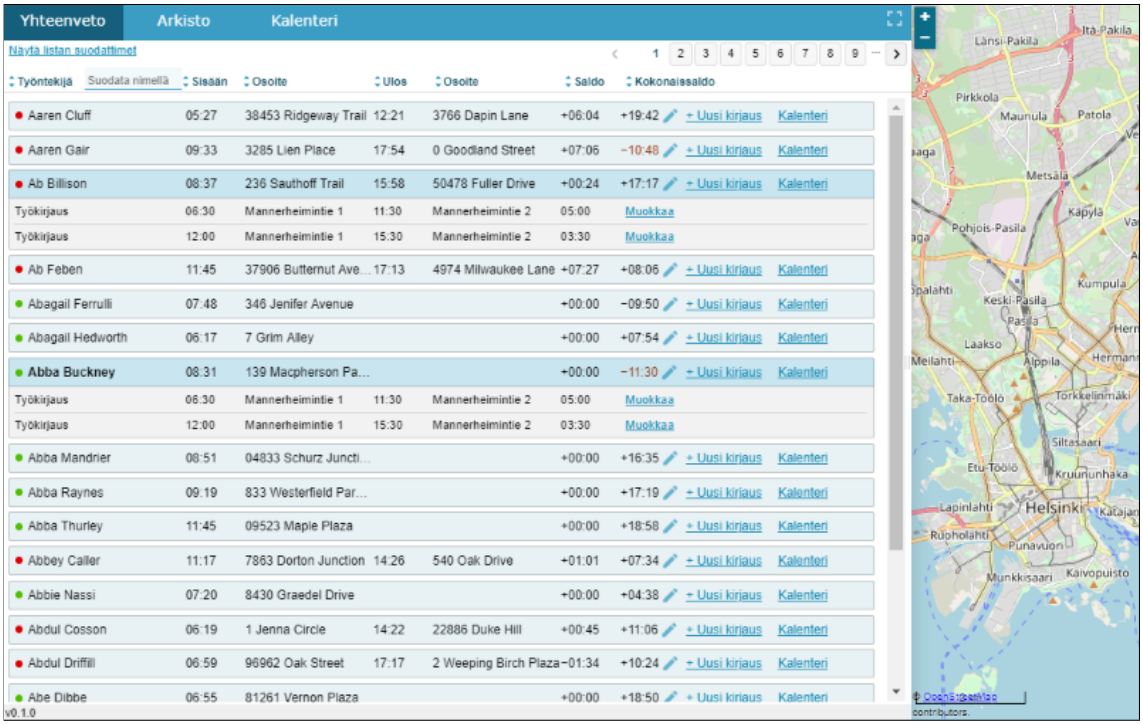
Kuten usein yksisivuisissa sovelluksissa, prototyypin tilan hallinta osoittautui monimutkaiseksi. Esimerkiksi osastohierarkia ja työntekijälista ovat tietoja, joiden pitäisi olla saatavilla sekä yhteenveto- että arkistonäkymissä. Koska tieto halutaan kuitenkin noutaa palvelimelta vain kerran, on se varastoitava johonkin niin, että se on kaikkien sitä tarvitsevien komponenttien saatavilla. Yksi ratkaisu olisi säilyttää tieto jollakin ylätason komponentilla, josta se kuljetetaan hierarkiassa alas päin tietyille alikomponenteille. Tietoa voi tällöin joutua kuljettamaan usean sellaisenkin komponentin läpi, jotka eivät itse tarvitse sitä, mikä monimutkaistaa monia komponenttimäärittelyitä turhaan. Prototyypissä päädyttiin ratkaisuun, jossa globaalit arvot varastoidaan erilliseen ”varasto-olioon”, johon voidaan viitata mistä tahansa komponentista käsin. Vue.js:n oma dokumentaatio on antanut ratkaisulle nimeksi ”store pattern” [vue19].

Vue.js tarjoaa käyttöliittymän tilan hallintaan myös valmiin ratkaisun, ”vuex”-kirjaston. Prototyypissä kuitenkin pyritään käyttämään mahdollisimman vähän riippuvuuksia, joten tavoitteena on selvitä toistaiseksi varasto-oliomallilla.

5.6 Yhteenvetonäkymä

Käyttöliittymäluonnoksissa ehdotettiin, että yhteenvetonäkymä olisi ryhmitelty osastohierarkian mukaan ja esitettäisiin sivutettuna, jotta massiivinen rivimäärä ei koituisi suorituskykyongelmaksi. Toteutusvaiheessa kävi kuitenkin ilmi, että hierarkkisen tiedon sivuttaminen ei ole yksinkertaista toteuttaa; rivien näkyvyyttä laskiessa on huomioitava työntekijärivien lisäksi myös osastorivit sekä kaikki alaosastojen ja edelleen näiden alaosastojen alla olevat rivit. Kymmenen rivin sivu voisi koostua esimerkiksi yhdestä osastosta, kahdesta tämän osaston työntekijästä, osaston alaosastosta, neljästä alaosaston työntekijästä ja toisesta alaosastosta, jonka työntekijärivit kuitenkin jäävät seuraavalle sivulle. Tässä tapauksessa sivutus rikkoo osaston ja työntekijän välisen yhteyden, jolloin hierarkkisesta näkymästä ei ole hyötyä.

Koska modernit web-käyttöliittymät tukevat yksitasoisia listoja huomattavasti paremmin, päädyttiin ratkaisuun, jossa yhteenvetonäkymä esitetään yhdellä tasolla osastohierarkian sijaan. Tietyn osaston työntekijöiden tarkastelu on mahdollista suodattamalla rivejä erillisen osastovalinnan mukaan. Valintalistaa ei tarvitse suodattaa tai sivuttaa, joten se on helppo esittää hierarkkisena. Prototyypin yhteenvetonäkymä on nähtävissä kuvassa 10.



Kuva 10: Prototyypin yhteenvetonäkymä

Käyttöliittymäluonnoksessa esitettiin, että osastohierarkian saisi näkyviin oikean ylälaidan ”Näytä osastohierarkia”-painikkeella. Koska osastohierarkia kuitenkin aukeaa sivun vasempaan laitaan, on loogisempaa, että sen esiin tuova painike on myös samalla suunnalla. Muutoin käy niin, että käyttäjän huomio ja toiminta kohdistuu ensin sivun oikeaan reunaan ja joutuu painikkeen klikkaamisen jälkeen hyppäämään vastakkaiselle laidalle, johon uusi elementti ilmestyy. Toteutetussa mallissa tätä ongelmaa ei ole, vaan katse ja osoitin pysyvät koko ajan vasemmassa reunassa. Elementin luonnetta eräänlaisena avattavana työkalupalkkina on korostettu lisäämällä siihen CSS-animaatio niin, että elementti vaikuttaa liukuvan esiin ja pois näkyvistä.

Näytettävän sivumäärän valinta siirrettiin samaan piilotettavaan elementtiin osastohierarkian kanssa, ja ”Näytä osastohierarkia” -painikkeen nimi muutettiin muotoon ”Näytä suodattimet”. Kyseessä on asetus, jota käyttäjät eivät oletettavasti muuta usein, joten ei ole mielekästä pitää sitä jatkuvasti näkyvillä. Osastohierarkiaelementin uudelleennimeäminen myös mahdollistaa sen, että samaan elementtiin voidaan jatkossa lisätä muitakin listan suodatusasetuksia.

Käyttöliittymäluonnoksessa on suunniteltu, että työntekijärivien avaaminen tapahtuisi nykyisen käyttöliittymän tavoin erillisellä ”Näytä kirjaukset” -painikkeella ei-

kä rivi kokonaisuudessaan siis olisi klikattava. Toteutuksessa kuitenkin päädyttiin siihen, että rivin voi avata mistä tahansa kohtaa klikkaamalla, pois lukien ”Uusi kirjaus”- ja ”Kalenteri”-painikkeet, joilla on omat toimintonsa. Tämä nopeuttaa kirjaustietoihin käsiksi pääsemistä, kun osoitinta ei tarvitse erikseen viedä oikeaan kohtaan riviä.

Yhteenvetolista on tehty horisontaalisesti responsiiviseksi määrittämällä sarakkeiden leveydet prosentteina, jolloin ne asettuvat selainikkunan koon mukaan lähemmäs tai kauemmas toisistaan. Mikäli lista käy niin kapeaksi, että sarakkeen teksti ei enää mahdu kokonaan näkyviin, teksti katkaistaan kesken ja päätetään kolmella pisteellä. Pystysuuntainen responsiivisuus on toteutettu lisäämällä listaan vierityspalkki.

5.7 Arkistonäkymä

Arkiston hakulomakkeen työntekijävalinta jouduttiin myös miettimään uusiksi suhteessa käyttöliittymäluonnoksiin, sillä osastot ja työntekijät sisältävän hierarkkisen, sivutetun listan esittäminen osoittautui mahdottomaksi. Muokkauslomakkeiden tavoin myös hakulomake toteutettiin vasemmassa reunassa olevana avattavana lomakkeena, joka on oletuksena auki, kun arkistonäkymään siirrytään ensimmäistä kertaa. Onnistuneen haun jälkeen lomake sulkeutuu tehdäkseen tilaa hakutuloslistalle. Lomakkeen saa takaisin näkyviin listan vasemman ylänurkan ”Näytä hakuehdot” -painikkeella.

Hakulomakkeella näytettävä työntekijälista hyödyntää samaa komponenttia kuin yhteenvetonäkymän lista. Avattavien työntekijärivien sijaan listassa näytetään työntekijöistä pelkästään nimet ja näitä vastaavat valintaruudut. Valintaruudun voi aktivoida klikkaamalla mitä tahansa kohtaa työntekijärivillä. Oletuksena yhtäkään työntekijää ei ole valittu, mikä nopeuttaa yksittäisten työntekijöiden kirjausten hakemista verrattuna nykyisen käyttöliittymän ratkaisuun, sillä valintaa ei tarvitse ensin kääntää. Lista sisältää yhteenvetonäkymän tavoin sivutuksen sekä suodatuksen työntekijän nimellä ja osastolla. Osastolistan saa näkyviin ”Näytä suodattimet” -painikkeella yhteenvetonäkymän tavoin.

Työntekijälista on jätetty melko lyhyeksi, vain kymmenen rivin korkuiseksi, jotta lomakkeen alareunan hakupainike ei jäisi liian kauas lomakkeen ylimmistä kentistä. Mikäli lomake olisi koko sivun korkuinen, joutuisi käyttäjä kuljettamaan osoitinta hyvin pitkän matkan suorittaakseen haun esimerkiksi päivämääräkenttien muuttamisen jälkeen.

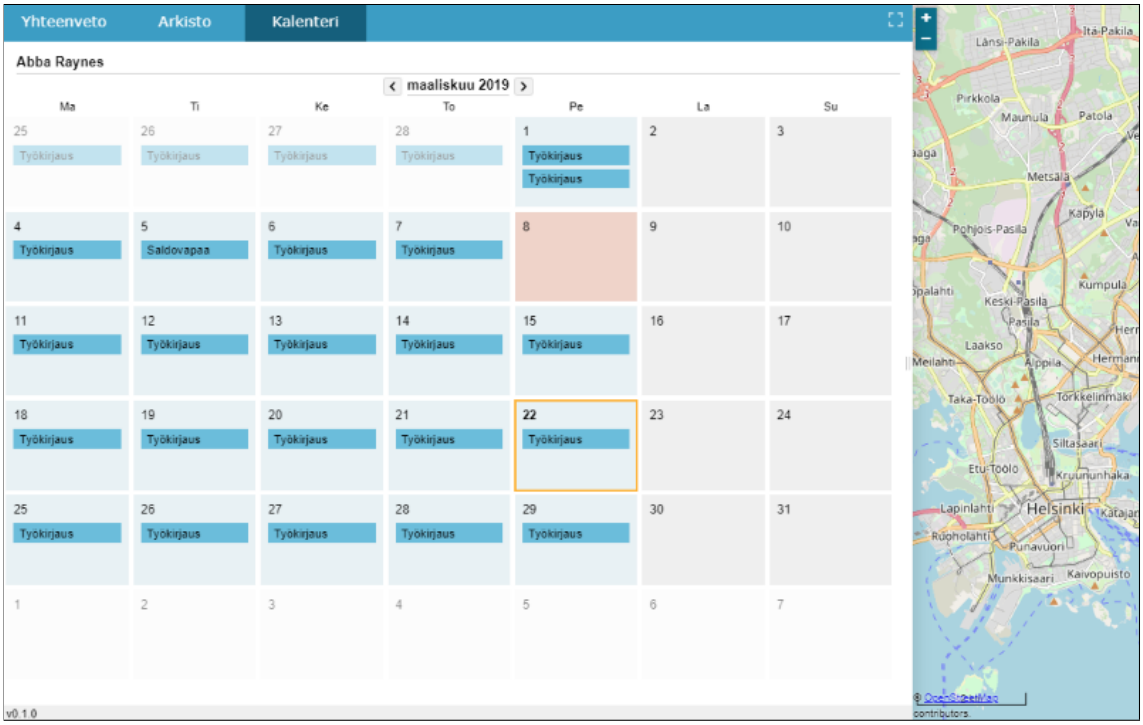
Myös arkiston hakutuloslista hyödyntää yhteenvetonäkymän kanssa samaa listakomponenttia ja muistuttaa sitä ulkonäöltään. Käyttöliittymäluonnoksissa ehdotettu ratkaisu eri työpäivien työaikakirjausten esittämiseen osoittautui toimivaksi; saman työpäivän aikana tehdyt kirjaukset näyttävät kuuluvan yhteen, ja käyttäjä säästää yhden klikkauksen, kun työpäivällä ei ole erillistä, avattavaa riviä. Työntekijän vapaapäivät on korostettu vaalealla ja kursiivilla fontilla, mikä auttaa hahmottamaan työviikkojen vaihtumisen. Lisäksi saldovapaat on erotettu normaaleista työaikakirjauksista vihreällä tekstillä ja puutteelliset päivät on puolestaan korostettu punaisella värillä.

5.8 Kalenterinäkymä

Kalenterinäkymä jätettiin työlistalla viimeiseksi, koska sitä ei ole nykyisessä käyttöliittymässä eikä se siksi ole ehdoton osa korvaavassa käyttöliittymässä. Komponentti kalenterin esittämiseen kuitenkin syntyi muita näkymiä tehdessä, sillä päivämääräkenttiä varten tarvittiin mahdollisuus valita päivä kalenterista. Kalenterikomponentti päädyttiin toteuttamaan itse valmiin komponenttikirjaston käyttämisen sijaan. Tämä mahdollistaa sen, että kalenterista saadaan muun käyttöliittymän visuaalisen tyylin mukainen ilman, että valmiin komponentin tyylejä tarvitsee ylikirjoittaa. Lisäksi itse tehdyn komponentin mukailtavuus on parempi, jolloin sitä voidaan hyödyntää helposti myös ”Kalenteri”-välilehdellä.

Käyttöliittymäluonnoksissa on esitetty, että myös kalenterinäkymässä olisi oma valintalista työntekijöille. Tämä tarkoittaisi sitä, että listakomponentista pitäisi tehdä vielä neljäs, hieman erilainen versio. Prototyypissä päädyttiin työmäärän vähentämiseksi ratkaisuun, jossa työntekijän kalenterin alustus tapahtuu vain yhteenvetonäkymän kautta, eli kalenterin työntekijälista korvataan yhteenvetonäkymän listalla. Kalenterinäkymässä ei siis ole mahdollista asettaa tai vaihtaa tarkasteltavaa työntekijää. Mikäli käyttäjä siirtyy välilehdelle suoraan, kalenteri jää tyhjäksi ja näkymässä näytetään ohjeteksti, että työntekijäkohtaiseen kalenteriin pääsee yhteenvetonäkymän kautta. Tämä ei todennäköisesti ole käyttökokemuksena miellyttävä, sillä käyttäjä joutuu siirtymään edestakaisin välilehtien välillä.

Prototyypin kalenterinäkymään, joka on nähtävissä kuvassa 11, toteutettiin ainoastaan päivän työaikakirjausten näyttäminen yksinkertaisina palkkeina sekä puutteellisten päivien korostaminen punaisella taustavärillä. Kirjausten lisääminen ja päivittäminen kalenterinäkymän kautta toteutetaan jatkokehityksen puitteissa, mikäli näkymä koetaan hyödylliseksi.



Kuva 11: Prototyypin kalenterinäkymä

5.9 Karttanäkymä

Osa kartan toiminnoista päätettiin toteutusvaiheessa rajata tämän työn ulkopuolella, sillä ne vaativat Gekko-karttakirjaston kehittämistä ja kasvattaisivat projektin työmäärää huomattavasti. Prototyypissä on käytössä karttakomponentista hyvin perusmallinen versio, jossa on tuki näkymän liikuttamiseen, zoomaustason vaihtamiseen sekä pistemäisten työaikaleimausten näyttämiseen. Ainoana karttakerroksena on avoimesta OpenStreetMap-karttapalvelusta saatava koko maailman kattava karttakerros, sillä se toimii missä tahansa käyttöympäristössä, missä on verkkoyhteys.

Käyttöliittymäluonnoksissa hahmoteltu lämpökarttakerros työaikakirjauksista vaatii myös enemmän suunnittelutyötä. Asiakastoiveena oli, että spatiaalisesti poikkeavat kirjaukset olisi helppo havaita, kun taas lämpökartta korostaisi erottuvimmilla sävyillä nimenomaan niitä alueita, joissa suurin osa kirjauksista on tehty. Parempi ratkaisu olisi siis jonkinlainen käänteinen lämpökartta, jossa pisteen väri olisi sitä voimakkaampi, mitä kauempana se on muista pisteistä. Tällöin spatiaalisesti muista poikkeavat pisteet näkyisivät kartalla esimerkiksi punaisen ja oranssin sävyillä, kun taas normaalit, runsaasti kirjauksia sisältävät alueet näkyisivät vihreänä tai sinisenä.

5.10 Päivitysnäkymät

Työaikakirjausten ja saldon muokkauslomakkeen esittämistapa oli prototyypin toteutuksen suurimpia ongelmakohtia. Nykyisen ulkoasun ja käyttöliittymäluonnosten modaalisten ikkunoiden sijaan päädyttiin kokeilemaan samankaltaista ratkaisua kuin osastohierarkian näyttämisessä: lomake tiedon syöttämiseen näytetään sivun vasemmassa reunassa. Ratkaisussa on se etu verrattuna modaalisiin ikkunoihin, että käyttäjä voi taustalla jatkaa yhteenvetonäkymän käyttöä ja esimerkiksi selata listaa eteenpäin sulkematta lomaketta. Huonona puolena ratkaisussa on se, että käyttäjän huomio joutuu hyppäämään oikealta vasempaan reunaan. Tämän takia voi olla, että käyttäjät eivät koe uutta lomakeratkaisua miellyttäväksi käyttää. Vasemman reunan kuitenkin todettiin olevan oikeaa reunaa parempi vaihtoehto, sillä oikeaan reunaan avautuva lomake peittäisi listan toimintopainikkeet. Tällöin käyttäjä ei voisi avata uutta lomaketta vanhan tilalle sulkematta edellistä lomaketta ensin.

Lomakkeiden kentissä on välitön tarkistus, eli virheellisesti tai puutteellisesti täytetyt kentät korostetaan varoitusvärillä. Lomakkeen tallentaminen on estetty, mikäli tiedot eivät kelpaa. Tietojen päivittäminen on sujuvampaa, kun käyttäjä saa heti palautetta virheistä.

Työaikakirjauksen lomakkeen kenttien järjestystä on pyritty muuttamaan loogisemmaksi. Ensimmäisenä valintana on kirjauksen tyyppi, sillä se on pakollinen ja oleellinen tieto. Sisään- ja ulosleimausten osoitekentät sijaitsevat heti leimausten kellonaika- ja päivämääräkenttien alla, kun taas nykyisessä käyttöliittymässä osoitteet ovat täysin erillään aikatiedoista. Sekä tietojen lukemisen että syöttämisen kannalta käytettävyys paranee, kun samaan asiaan liittyvät tiedot sijaitsevat lähellä toisiaan.

Leimaustietojen kentät on nimetty samaan tapaan kuin yhteenvetolistan sarakkeet: ”Sisään”, ”Osoite”, ”Ulos” ja ”Osoite”. Kaksi samannimistä kenttää saattavat hämmentää käyttäjää, vaikka niiden asettelusta voikin päätellä, että ensimmäinen osoitekenttä viittaa sisäänleimaukseen ja jälkimmäinen ulosleimaukseen. Vaihtoehtoisista nimistä esimerkiksi ”Sisäänleimauksen osoite” olisi melko pitkä ja ”Saapumisosoite” puolestaan ristiriidassa sen kanssa, mitä termejä muualla sovelluksessa käytetään.

Kirjauksen poistopainike, joka nykyisessä käyttöliittymässä sekä käyttöliittymäluonnoksessa sijaitsi suoraan kirjauksen rivillä, on prototyypissä sijoitettu muokkauslomakkeelle. Kyseessä on peruuttamaton ja todennäköisesti harvoin käytettävä toiminto, joten sen ei tarvitse olla helposti saavutettavissa. Lomakkeella ”Poista”-painike on

sijoitettu tallennus- ja peruutuspainikkeiden yhteyteen, mutta se on niitä pienempi ja tyyliltään erilainen.

5.11 Asettelu ja värit

Prototyypissä on pyritty Gestaltin hahmolakeja noudattavaan yhdenmukaisuuteen samantyyppisten elementtien välillä, mitä tukee se, että elementit hyödyntävät samoja määrityksiä JavaScript-koodissa. Yhteenveto-, hakutulos- ja valintalistoille yhteiset toiminnot, kuten suodattaminen ja sivutus, näyttävät samalta ja sijaitsevat jokaisessa näkymässä samassa paikassa. Lomakkeiden sulkemispainike sijaitsee aina oikeassa ylänurkassa. Lomakkeen pääasiallinen toimintopainike – tietojen syöttölomakkeella ”Tallenna” ja hakulomakkeella ”Hae työaikakirjauksia” – näyttää samalta joka lomakkeella. Kentissä, joihin käyttäjä voi syöttää tietoa, on kaikilla lomakkeilla sama tyyli riippumatta siitä, onko kyseessä valintalista, päivämäärävalinta vai yksinkertainen tekstikenttä. Listan työntekijärivit ovat väriltään, kooltaan ja muodoltaan samanlaiset yhteenvetonäkymässä ja hakutuloksissa.

Elementtikokonaisuudet on erotettu muista elementeistä erotinviivoilla tai jättämällä tyhjä alue elementtien väliin. Esimerkiksi työntekijärivien välissä on tyhjää tilaa, kun taas työntekijään liittyvät kirjausrivit ovat kiinni työntekijärivissä. Hakulomakkeen työntekijävalintaan liittyvät elementit, kuten sivutuspainikkeet, on rajattu reunaviivoilla ja taustavärillä omaksi alueekseen. Lomakkeiden kenttien selitetekstit on tasattu oikealle eli kenttien viereen, jotta läheisyyden hahmolaki auttaa käyttäjää hahmottamaan, mikä selite vastaa mitäkin kenttää.

Kuva 12: Listan sivun valinta prototyypissä

Esimerkiksi kuvassa 12 näkyvästä listan sivutuselementistä voidaan tunnistaa seuraavat digitaalisen median Gestaltin hahmolait [Eva17], jotka auttavat hahmottamaan painikkeet osana samaa kokonaisuutta:

- painikkeet sijaitsevat vain muutaman pikselin päässä toisistaan
- epäaktiivisilla painikkeilla on sama tausta-, reuna- ja fonttiväri

- painikkeet ovat saman muotoisia ja kokoisia
- painikkeet sijaitsevat samalla horisontaalisella janalla.

Käyttöliittymän värimaailma yhdistelee valkoista, vaaleanharmaata ja vaaleansinisen eri sävyjä, sillä nämä värit ovat käytössä myös yrityksen tuoteperheen muissa sovelluksissa. Sinistä käytetään pääasiassa sellaisissa elementeissä, joita klikkaamalla tapahtuu jotain, esimerkiksi työntekijäriveissä ja painikkeissa. Yleisperiaatteena on, että mustaa tekstiä napsauttamalla ei tapahdu mitään. Korostusväreinä käytetään punaista ja oranssia. Punainen toimii varoitusvärinä lomakkeiden poistopainikkeissa, virheellisissä syötteissä, virheilmoituksissa, puutteellisissa päivissä sekä varoitusrajan alittavissa saldomäärissä. Keltaoranssia taas käytetään korostamaan aktiivisia kohteita, kuten muokattavaa työntekijäriviä tai valittua päivää kalenterissa. Kaikki valitut värisävyt ovat murrettuja, jotta ne eivät rasittaisi silmää.

Listan rivien toiminnot, kuten ”Uusi kirjaus” ja ”Kalenteri”, on toteutettu mukaillen hyperlinkkien yleistä oletustyyliä eli sinistä tekstiä alleviivauksella. Kolmiulotteiset, värikkäät painikkeet tekisivät listan oikeasta reunasta visuaalisesti raskaan ja turhan huomiota herättävän. Tekstimäinen elementti myös sopii paremmin kapeaan tilaan kuin painike, joka vaatisi enemmän tyhjää tilaa ympärilleen. Käyttäjällä todennäköisesti on hyperlinkeistä mentaalinen malli, joka auttaa hahmottamaan nämä elementit klikattavina.

Listan rivielementin klikattavuus on tuotu esille valkoisesta taustasta erottuvalla taustavärillä sekä sillä, että elementin taustaväri ja osoittimen ulkonäkö muuttuvat, kun osoittimen vie elementin päälle. Varjostusta ei ole käytetty kolmiulotteisuuden vaikutelman tuomiseen, sillä listan ulkoasu on haluttu pitää mahdollisimman selkeänä, jotta liika tyylittely ei vaikeuttaisi luettavuutta. Varjostus jokaisessa rivissä saattaisi viedä käyttäjän huomiota pois olennaisesta. Sen sijaan varjostusta on käytetty korostamaan lomakkeiden painikkeita. Tallennus- ja hakupainikkeissa on myös muita elementtejä tummempi taustaväri sekä poikkeuksellisesti valkoinen teksti. Tärkeät painikkeet saadaan näin erottumaan vahvasti lomakkeiden muista elementeistä sekä toissijaisista lomaketoiminnoista, kuten ”Peruuta”-painikkeesta. Tämä auttaa käyttäjää löytämään halutun painikkeen nopeasti silloinkin, kun käyttäjän mentaalisessa mallissa vahvistus- ja peruutuspainikkeet ovat eri päin kuin prototyypissä.

5.12 Ikonit ja symboliikka

Prototyypissä käytettävät ikonit on enimmäkseen haettu Googlen kehittämästä, avoimesta Material Design -valikoimasta, jonka käytöstä on sovittu yrityksessä. Ikonit ovat tyyliltään yksinkertaisia ja tehty toimimaan pienessäkin koossa, mikä on välttämätöntä, jotta ikonit saadaan mahtumaan esimerkiksi listan riveille. Osaa valmiista ikoneista on muokattu prototyyppiä varten, jotta niistä on saatu paremmin käyttötarkoitukseen sopivia.

Mahdollisuus järjestää lista sarakkeen mukaan sarakeotsikkoa klikkaamalla on ilmaistu otsikon vieressä olevalla ikonilla, jossa on ylös- ja alaspäin osoittavat nuolet. Selkeintä olisi, jos ikoni muuttuisi sen mukaan, määrääkö kyseinen sarake järjestyksen ja onko järjestys nouseva vai laskeva. Tämän toteuttaminen ei kuitenkaan ole täysin triviaalia, sillä tieto järjestyksen määräävästä sarakkeesta ei ole listan otsikkokomponentilla vaan sen vanhemmalla, johon otsikkokomponentti kuljettaa vain tiedon siitä, mitä sarakeotsikkoa on klikattu. Jatkokehitykseksi jää se, että vanhempikomponentilta annetaan listan otsikolle tieto päivittyneestä järjestyksestä.

Saldon muokkauslomake aukeaa painikkeella, jossa on kynää esittävä ikoni. Kynäsymboli on hyvin pelkistetty eikä yhtä selkeä kuin nykyisen käyttöliittymän monivärinen ja yksityiskohtaisempi muokkausikoni. Painike ei muutenkaan ole yhtä ymmärrettävä kuin käyttöliittymän muut, hyperlinkkiä tai kolmiulotteista nappia muistuttavat painikkeet, joissa elementin teksti kertoo suoraan, mitä sen klikkaaminen tekee. Heikompi ymmärrettävyys on kuitenkin perusteltavissa sillä, että kyseessä on toiminto, jota pitäisi joutua käyttämään vain hyvin harvoin ja jonka vain harvat käyttäjät näkevät. Muokkauspainiketta on pyritty selkeyttämään sijoittamalla se aivan kokonaissaldoarvon viereen, kuten nykyisessäkin käyttöliittymässä, ja värjäämällä ikoni samalla sinisellä, joka on käytössä muissa toimintopainikkeissa. Ikonin merkitystä avataan siis läheisyyden ja samankaltaisuuden hahmolailla sekä mukailemalla mentaalisia malleja nykyisestä käyttöliittymästä ja siitä, miltä klikattavat asiat näyttävät uudessa käyttöliittymässä.

Pystysuora alue, jota vetämällä käyttäjä voi muuttaa käyttöliittymän lohkojen suhdetta, on merkattu ikonilla, jossa on kaksi lyhyttä, vierekkäistä viivaa. Ikoni mukailee sitä, miltä osoitin näyttää Windows-käyttöjärjestelmässä, kun sen kohdalla olevan elementin kokoa voi muuttaa vetämällä.

Karttanäkymän piilottavaan painikkeeseen ei löytynyt itsestäänselvää vaihtoehtoa Material Designin ikonien joukosta. Ikonilla voitaisiin viestiä joko kartan piilotta-

mista tai sitä, että käyttöliittymän vasen lohko venytetään koko ruudun levyiseksi. Koska painike sijaitsee vasemmassa lohkossa eikä kartassa, on loogisempaa, että ikoni symboloi jotakin asiaa, joka tehdään nimenomaan vasemmalle lohkolle. Näin ollen ikoniksi valittiin symboli, joka kuvastaa ikkunan avaamista suuremmaksi. Listan järjestyspainikkeiden tavoin myös tämän painikkeen ikonin pitäisi muuttua sen mukaan, onko kartta näkyvissä vai ei. Ikonin tilan päivittäminen jää jatkokehitykseksi tämänkin painikkeen osalta.

Aina, kun selain odottaa vastausta palvelimelta – esimerkiksi työntekijätietoja haettaessa – näytetään käyttöliittymässä latausindikaattori. Indikaattori on pyörivä ympyrä, joka on valikoitu ”loading.io”-sivuston ilmaisten latausikonien valikoimasta. Nykyisessä käyttöliittymässä ainoa vastaava indikaattori on haun yhteydessä, jolloin näytetään vain staattinen ”Haku käynnissä...”-teksti. Liikkuva ikoni tekee käyttöliittymästä eloisamman ja viestii käyttäjälle paremmin, että jotain on käynnissä eikä sovellus tai selain ole jumiutunut. Ikoni on myös kontekstiriippumaton, joten se toimii jokaisen asynkronisen kutsun yhteydessä.

6 Prototyypin arviointi

Tässä luvussa tarkastellaan, miten valmis prototyyppi toteuttaa sille suunnitteluvaiheessa määritellyt vaatimukset ja miten hyvä sen käytettävyys on. Vaatimusmäärittelyssä esitettiin, että prototyypin pitää tarjota ratkaisut käyttötapauksiin ja nykyisen käyttöliittymän ongelmakohtiin sekä tukea työpöytä- ja tablettikäyttöä suosituimmilla selaimilla. Lisäksi lähdekoodin laatu on todennettava staattisella analyysillä ja kattavalla automaatiotestauksella. Lopuksi luvussa arvioidaan prototyypin käytettävyyttä pienimuotoisella käytettävyystestauksella.

6.1 Käyttötapaukset

Kuvassa 3 esitetyt vaatimusmäärittelyn käyttötapaukset on kirjoitettu auki alla olevaan taulukkoon 6. Keskimmäisessä sarakkeessa on kuvattu käyttötapausta vastaava ratkaisu sellaisena, kuin se on prototyypissä toteutettu. X-kirjain oikeimmanpuoleisessa sarakkeessa kertoo, että prototyyppi toteuttaa käyttötapauksen riittävällä tasolla. Esimerkiksi kirjausten raportoinnin tapauksessa tämä tarkoittaa sitä, että painikkeelle on olemassa paikka ja ulkoasu, vaikka varsinaista raportin palauttavaa logiikkaa ei ole toteutettu. Erityisoikeuksia vaativissa käyttötapauksissa todettiin, että kooditoteutuksen sijaan riittää, että toimintojen näkyvyyden rajaaminen on suunniteltu.

Käyttötapauksista jäi toteuttamatta ainoastaan spatiaalisesti poikkeavien kirjausten esittämisen tapa. Ominaisuuden todettiin tarvitsevan vielä lisää suunnittelua ennen kuin sitä on järkevä lähteä toteuttamaan. Ominaisuutta ei myöskään ole olemassa nykyisessä käyttöliittymässä, joten se ei ole välttämätön uuden käyttöliittymän käyttöönoton kannalta. Pääasiassa prototyypissä onnistuttiin kattamaan määritellyt käyttötapaukset hyvin.

Käyttötapaus	**Prototyypin ratkaisu**	**OK**
Esimies näkee kuluvan päivän työaikakirjausten yhteenvedon.	Yhteenveto on oma välilehtensä kuten nykyisessä käyttöliittymässä.	X
Esimies voi hakea työaikakirjauksia päivämäärävälillä.	Arkiston hakulomakkeella on alku- ja loppupäivän valinta.	X
Esimies voi hakea työaikakirjauksia työntekijöittäin.	Arkiston hakulomakkeella on suodatettava lista työntekijöistä, joista voi valita yhden tai useamman.	X

Esimies voi hakea puutteelliset työpäivät.	Arkiston hakulomakkeella on valintaruutu ”Vain puutteelliset päivät”.	X
Esimies pystyy erottamaan hakutuloksista ajallisesti poikkeavat kirjaukset.	Arkiston hakutuloslistassa sekä kalenterissa on korostettuna varoitusvärillä päivät, jotka sisältävät puutteellisia tai poikkeavia kirjauksia.	X
Esimies pystyy erottamaan hakutuloksista spatiaalisesti poikkeavat kirjaukset.	–	–
Esimies voi tuottaa raportin työaikakirjauksista tietyillä hakuehdoilla.	Arkiston hakulomakkeella on ”Hae Exceliin” -painike.	X
Esimies voi katsella työaikakirjauksia.	Työaikakirjauksia voi tarkastella yhteenvetonäkymän, arkiston ja kalenterin kautta.	X
Esimies voi katsella työntekijän saldoa.	Työntekijän päivä- ja kokonaissaldo ovat nähtävissä yhteenvetonäkymässä.	X
Esimies voi katsella työaikakirjauksen muutoshistoriaa.	Kirjauksen muutoshistoria on nähtävissä sen muokkauslomakkeella.	X
Esimies voi katsella saldon muutoshistoriaa.	Saldon muutoshistoria on nähtävissä sen muokkauslomakkeella.	X
Erityisoikeuksilla voi lisätä uuden työaikakirjauksen.	Yhteenvetonäkymässä ja hakutuloksissa on ”Uusi kirjaus” -painikkeet, jotka on piilotettu peruskäyttäjiltä.	X
Erityisoikeuksilla voi poistaa työaikakirjauksen.	Kirjauksen muokkauslomakkeella on ”Poista”-painike, joka on piilotettu peruskäyttäjiltä.	X
Erityisoikeuksilla voi muokata työaikakirjausta.	Kirjausrivillä on ”Muokkaa”-painike, joka avaa päivityslomakkeen. Peruskäyttäjillä ”Muokkaa”-painikkeen nimi on ”Näytä lisätiedot” ja lomakkeen ”Tallenna”-painike on estetty.	X
Erityisoikeuksilla voi muokata työntekijän saldoa.	Yhteenvedon työntekijärivillä on kokonaissaldoarvon vieressä painike, joka avaa muokkauslomakkeen. Peruskäyttäjillä lomakkeen ”Tallenna”-painike on estetty.	X

Taulukko 6: Prototyypin ratkaisut käyttötapauksiin

6.2 Ratkaisut ongelmakohtiin

Alla olevaan taulukkoon 7 on koottu alaluvussa 4.2 havaitut ongelmat nykyisessä käyttöliittymässä. Oikeanpuoleisessa sarakkeessa on kuvattu prototyyppiin toteutettu ratkaisu ongelmalle. Ongelmakohtia löydettiin yhteensä 17, ja kaikkiin näihin on prototyypissä vastattu jollakin tavalla.

Nykyinen ongelma	**Prototyypin ratkaisu**
Käyttöliittymä ei ole responsiivinen. Selailu on vaikeaa kapealla näytöllä tai mobiililaitteella.	Käyttöliittymä mukautuu selainikkunan leveyden mukaan.
Käyttöliittymä näyttää vain yhden osaston työntekijät eikä tue alaosastojen näyttämistä.	Työntekijätietoja voi tarkastella esimiehen kaikista osastoista.
Valintaruutujen aktivointi on hankalaa, sillä se onnistuu vain laatikkoa itseään klikkaamalla.	Valintaruudun voi aktivoida myös klikkaamalla sen viereistä tekstiä.
Tietyn työntekijän tietojen löytäminen on vaikeaa pitkästä listasta.	Työntekijälistan voi suodattaa työntekijän nimellä ja järjestää sarakkeittain.
Saman näköiset kolmiulotteiset rivielementit ovat välillä auki klikattavia ja välillä eivät.	Auki klikattavat työntekijärivit näyttävät samalta sekä yhteenveto- että arkistonäkymässä. Elementtien klikattavuus tuodaan esiin värivalinnoilla ja muuttamalla elementin ulkoasua osoittimen ollessa sen kohdalla.
Kirjauksia voi hakea korkeintaan kuukauden aikaväliltä.	Kirjauksia voi hakea miltä tahansa aikaväliltä. Palvelinpää hoitaa mahdolliset raskaan haun rajoitukset.
Keskeneräisen haun indikaattori on staattinen teksti eikä täten ole erityisen mielenkiintoista katsottavaa odotellessa.	Latausindikaattori on liikkuva kuva.
Hakutulokset nähdäkseen näkymää joutuu vierittämään alaspäin, mikäli hakulomakkeen työntekijälista on pitkä.	Hakulomake sijaitsee hakutuloslistan vieressä, ja se piilotetaan haun valmistuttua.
Tietyn kirjauksen löytäminen hakutuloksista on työlästä, sillä se vaatii sekä oikean työntekijärivin että päivärivin auki klikkaamisen.	Työaikakirjaukset sijaitsevat suoraan työntekijärivin alla.

Modaaliset muokkauslomakkeet eivät näytä, kenen työntekijän tietoja ollaan käsittelemässä.	Muokkauslomakkeilla on työntekijän nimi. Lisäksi lomakkeen työntekijä näkyy korostettuna listassa.
Muokkauslomakkeen kentät on asettelun takia vaikea hahmottaa.	Kenttien selitteet on tasattu oikealle, kenttien viereen. Lomakkeen rivien välissä on tyhjää tilaa erottimena.
Muokkauslomake ei anna välitöntä palautetta virheellisestä kellonajasta tai pakollisista kentistä.	Virheellisesti täytetty kenttä korostetaan välittömästi punaisella värillä.
”Tallenna”-painike sijaitsee ”Sulje”-painikkeen oikealla puolella, mikä on ristiriidassa tuoteperheen muiden sovellusten ja Windows-käyttäjien tottumusten kanssa.	Muokkauslomakkeen ”Peruuta”-painike sijaitsee tallennuspainikkeen oikealla puolella.
”Tallenna”- ja ”Sulje”-painikkeet näyttävät samalta, jolloin oikeaa toimintoa ei löydä lukematta painikkeiden nimiä.	”Peruuta”-painike on tausta- ja fonttiväriltään sekä tummuusasteeltaan erilainen kuin tallennuspainike. ”Tallenna”-painike on ulkoasultaan huomiota herättävämpi.
Kokonaissaldolle ja kirjauksille aiemmin tehdyt muutokset eivät ole nähtävissä.	Kirjauksen ja saldon muokkauslomakkeella näkyy muutoshistoria.
Sovelluksen näkymien tila ei säily välilehdeltä toiselle siirryttäessä.	Välilehdet säilyttävät tilansa.
Kartta alustuu aina mittakaavaan, jossa näkyy koko Suomi.	Kartta muistaa evästeiden avulla käyttäjän aikaisemman keskityksen ja zoomauskertoimen.

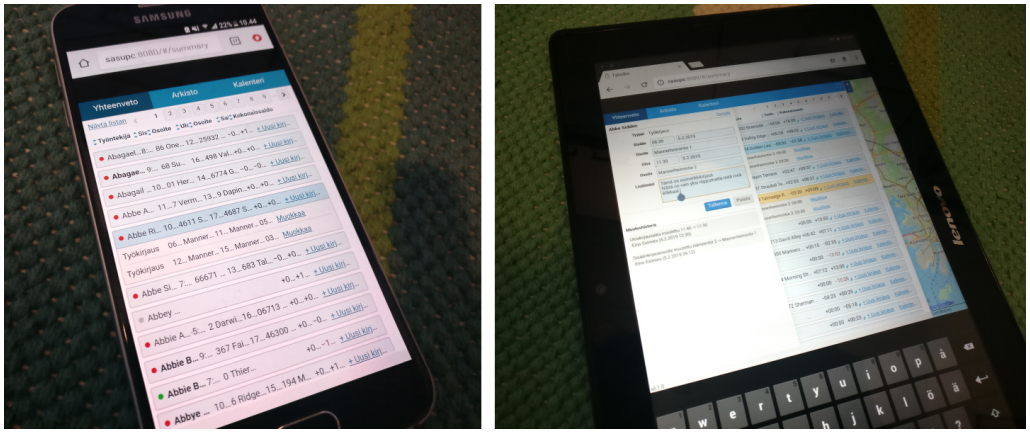
Taulukko 7: Prototyypin ratkaisut nykyisiin ongelmiin

6.3 Mukautuvuus

Vaatimusmäärittelyn mukaan prototyypin on toimittava sekä työpöytälaitteella että tabletilla. Tabletilla testaaminen suoritettiin Lenovo A7600-F -merkkisellä laitteella, jossa on 10,1-tuumainen näyttö ja käyttöjärjestelmänä Android 4.4.2.

Prototyyppi on testilaitteella käyttökelpoinen sekä vaaka- että pystyasennossa, mutta käytettävyys ei ole yhtä hyvä kuin työpöytälaitteella. Osastohierarkian valintaruutujen laittaminen päälle ja pois päältä on heikommasta tarkkuudesta johtuen hankalampaa kosketusnäytöllä kuin hiirellä. Myös vasteajat valintaruudun tilan päi-

vittymisessä sekä esimerkiksi listan sivulta toiselle siirtymisessä ovat pitkiä, mutta tämä todennäköisesti johtuu jo noin viisi vuotta vanhan laitteen heikosta suorituskyvystä.



Kuva 13: Prototyyppi älypuhelimella ja tabletilla

Tietojen syöttäminen lomakkeille on helpompaa laitteen ollessa pystyasennossa, sillä vaaka-asennossa näyttönäppäimistö vie huomattavan osuuden näytön korkeudesta. Kuvassa 13 näkyy kirjauksen syöttölomake avattuna pystyasennossa olevalla tabletilla.

Toimivuus tabletilla testattiin onnistuneesti Google Chrome- ja Firefox-selaimilla. Laitteen omalla selaimella käyttöliittymän alustus kuitenkin epäonnistuu kokonaan. Päätettiin, että ongelman selvittämiseen ei kannata käyttää aikaa tässä vaiheessa.

Prototyypin käyttöä kokeiltiin myös älypuhelimella, sillä vaatimusmäärittelyn mukaan käyttöliittymän olisi hyvä olla jossain määrin käytettävä kännykällä. Listojen luettavuus on pystyasennossa huono, sillä kullekin sarakkeelle jää hyvin vähän tilaa leveyssuunnassa. Esimerkiksi saldon muokkauspainike jää kokonaan piiloon. Uuden kirjauksen lisääminen ja arkiston hakuehtojen asettaminen kuitenkin onnistuvat älypuhelimella yhtä sujuvasti kuin työpöytälaitteella.

On todennäköistä, että listojen sarakkeiden näkyvyys muodostuu ongelmaksi myös tabletilla näyttökoosta riippuen. Yhtenä ratkaisuna olisi piilottaa vähemmän tärkeitä sarakkeita kokonaan sitä mukaa, kun näytön leveys pienenee. Tällöin kapealla näytöllä yhteenvetolistassa näkyisivät esimerkiksi vain työntekijän nimi ja sisään- ja ulosleimausten kellonajat.

6.4 Vasteajat eri selaimilla

Prototyypin vasteaikoja arvioitiin mittaamalla eri operaatioiden suorittamiseen kuluvaa aikaa. Operaatioiksi valittiin yhteenvetonäkymän alustus, yhteenvetolistan sivulta toiselle siirtyminen sekä kaikkien työntekijärivien valitseminen ja valinnan kääntäminen arkistonäkymän hakulomakkeella. Nämä operaatiot ovat sellaisia, jotka todennäköisimmin ovat raskaita suurella tietomäärällä. Jokainen operaatio toistettiin viisi kertaa neljällä eri selaimella, ja tästä saadut keskiarvot ja -hajonnat on koottu taulukkoon 8. Arvot ovat kymmenen millisekunnin tarkkuudella.

Käyttöliittymän alustukseen kuluva aika on laskettu JavaScriptissä ”Window.performance”-rajapinnan avulla. Karttaruutujen hakemiseen kuluvaa aikaa ei ole laskettu mukaan alustuksen kestoon. Muut operaatiot on mitattu selainten omien kehittäjätyökalujen ”Suorituskyky”-välilehdellä. Operaatioiden kestoa havainnoitiin myös silmämääräisesti.

operaatio	**Chrome**	**Firefox**	**Edge**	**IE 11**
käyttöliittymän alustus	990(60)	1 850(150)	2 755(60)	2 870(70)
listan sivulta toiselle siirtyminen	140(20)	240(10)	440(20)	1 510(10)
5 000 rivin valitseminen	100(10)	130(0)	190(20)	1 160(70)
5 000 rivin valinnan peruminen	1 340(10)	60(0)	130(20)	250(40)

Taulukko 8: Prototyypin eri operaatioiden suorituskyky eri selaimilla

Chromella käyttöliittymän alustus tapahtuu keskimäärin alle sekunnissa, kun taas muilla selaimilla se kestää vajaasta kahdesta sekunnista lähes kolmeen sekuntiin. Lisäksi muilla selaimilla on havaittavissa latausindikaattorin hetkellinen pysähtyminen, mikä johtuu käyttöliittymän jumiutumisesta.

Myös listan sivun vaihtaminen sekä kaikkien työntekijärivien valitseminen tapahtuvat nopeiten Chromella, mutta 5 000 työntekijärivin valinnan perumisessa se suoriutuu muita selaimia huomattavasti heikommin; siinä missä Firefoxilla valintojen peruminen tapahtuu alle 0,1 sekunnissa, Chromella siinä kuluu yli sekunti. Internet Explorerilla listan sivulta toiselle siirtyminen sekä kaikkien rivien valinta kestävät myös yli sekunnin, mikä on se kynnys, jonka ylittävät prosessit tarvitsisivat latausindikaattorin [Nie94].

Silmämääräisesti arvioituna kaikki vasteajat ovat huomattavasti parempia, mikäli työntekijöitä on viidentuhannen sijaan vain tuhat. Esimerkiksi käyttöliittymän alustus tapahtuu tällöin kaikilla selaimilla lähes yhtä sujuvasti. Viidellätuhannella-

kin työntekijällä monien operaatioiden kesto pysyy noin 0,1–0,2 sekunnissa, jolloin käyttäjä kokee järjestelmän reagoivan välittömästi [Nie94]. Käyttöliittymän alustukseen kuluva aika sekä Chromella esiintyvä viive valittujen rivien perumisessa voivat kuitenkin muodostua ongelmaksi.

Prototyypin heikommat vasteajat Internet Explorerilla on perusteltavissa sillä, että kyseessä on vähitellen käytöstä poistuva selain. Microsoftin virallisessa blogissa helmikuussa 2019 julkaistun kirjoituksen mukaan Internet Explorer on nykyään lähinnä yhteensopivuusratkaisu, jota ei kehitetä tukemaan uusimpia ominaisuuksia ja jota useat kehittäjätkään eivät enää käytä testaamisessa [Jac19].

Käyttöliittymän toimivuus testattiin vain Android- ja Windows-käyttöjärjestelmillä. Applen macOS- ja iOS-käyttöjärjestelmät jäivät testaamatta sopivien laitteiden puutteessa, vaikka vaatimusmäärittelyssä esitettiin, että prototyypin olisi toimittava myös Safari-selaimella. Toimivuuden kartoittaminen Apple-laitteilla ja mahdollisten puutteiden korjaaminen jäävät siis jatkokehitykseen.

6.5 Koodin ylläpidettävyys

Lähdekoodi on Jest-testauskirjaston kattavuustyökalun mukaan 100-prosenttisesti testattua. Työkalun tuottama raportti automaatiotesteistä on nähtävissä liitteessä 2. Täyden testikattavuuden saamiseksi osa funktioista on määrätty sivuutettavaksi kattavuutta laskiessa. Näitä metodeja on yhteensä kahdeksan, joista suurin osa on tyhjiä funktioita. Nämä funktiot ovat koodin toimivuuden kannalta oleellisia mutta eivät sisällä mitään testaamista vaativaa. Lisäksi kattavuuden laskenta ohittaa funktion, joka selvittää selaimen käyttäjäagenttia tarkastelemalla, onko sovelluksen käyttäjällä mobiililaite. Kyseisessä metodissa on niin monta eri haaraa, ettei ole mielekästä kirjoittaa testejä kuin osalle tapauksista.

Kattava yksikkötestaus edesauttaa sitä, että käyttöliittymään on helpompi tehdä muutoksia ilman, että jokin osa siitä hajoaa huomaamatta. On kuitenkin huomioitava, että 100-prosenttinen testikattavuuskaan ei tarkoita sitä, että testit olisivat laadukkaita tai hyödyllisiä – kattavuus on vain kvantitatiivinen mittari. Automaatiotestaus ei myöskään korvaa manuaalista, ihmisen tekemää testausta.

Koodi pidetään selkeänä ja luettavana ESLint-analyysityökalulla. Mikäli ESLint havaitsee virheen, prototyypin kääntäminen epäonnistuu. Monet koodin luettavuutta vaikeuttavat asiat, kuten funktioiden epälooginen nimeäminen, jäävät kuitenkin automaattiselta työkalulta havaitsematta. Tämän vuoksi on tärkeää, että koodi on

myös manuaalisesti vertaisarvioitua. Prototyyppivaiheessa vertaisarviointia ei tehty resurssien vähyydestä johtuen. Tämä mahdollisti nopeatempoisemman kehityksen, mutta todennäköisesti hankaloittaa muiden kehittäjien osallistumista jatkokehitykseen.

6.6 Käytettävyystestaus

Käytettävyystestauksessa käytetty ohje löytyy liitteestä 3. Testaajiksi valittiin kolme henkilöä, joille sovellus on ennestään tuntematon. Testiryhmä poikkeaa näin ollen käyttöliittymän todellisista loppukäyttäjistä, joilla on todennäköisesti kokemusta nykyisestä sovelluksesta. Kaikki testaajat ovat kokeneita verkkosovellusten käyttäjiä, mutta heillä ei ole juurikaan aikaisempaa kokemusta työajanseurannan hallintajärjestelmistä. Testaus suoritettiin kannettavalla tietokoneella Google Chrome -selaimella tämän tutkielman tekijän toimiessa testaussession valvojana, ohjeistajana ja kirjurina. Tehtävien suorituksen aikana koneen kuvaruutua nauhoitettiin selaimen Screencastify-lisäosalla myöhempää tarkastelua varten.

Kukin testaussessio koostui viidestä lyhyestä tehtävästä, joiden tarkoituksena on selvittää, miten hyvin testaaja ymmärtää käyttöliittymää ja onnistuu löytämään tarvittavan tiedon tai toiminnon. Ensimmäisessä vaiheessa testaajan annettiin tutustua käyttöliittymään pintapuolisesti. Koska kyseessä on työkäyttöön tarkoitettu toimistosovellus, ei ole syytä olettaa, että sitä pitäisi pystyä käyttämään saman tien ilman minkäänlaista aikaisempaa kokemusta tai koulutusta. Alun tutustumisen jälkeen testin valvoja käynnisti kuvaruudun nauhoituksen ja alkoi kirjata muistiinpanoja testaajan toiminnasta. Testaajaa kehotettiin ajattelemaan ääneen.

Toisessa tehtävässä pyydetään lisäämään kuluvalle päivälle työkirjaus tietyn nimiselle työntekijälle. Tehtävä testaa kirjauksen syöttölomakkeen käytettävyyttä sekä sitä, miten testaaja onnistuu hakemaan halutun työntekijän 5 000 rivin joukosta. Kolmas tehtävä on puhdas tiedonhakutehtävä, jossa testaajan on osattava suodattaa yhteenvetolistalla näkyviin vain yhden osaston työntekijät ja järjestettävä nämä rivit tietyn sarakkeen mukaan.

Neljäs tehtävä kattaa saldon muokkauslomakkeen ja vaatii myös tiedon suodattamista. Testaajan on osattava hakea työntekijä osaston, nimen osan sekä kirjauksen tilan perusteella: ”Metsäosastossa” on yhteensä kolme työntekijää, joiden nimi täsmää tehtävänantoon, mutta vain yhdeltä heistä puuttuu kuluvalta päivältä ulosleimaus.

Viimeisessä tehtävässä testataan arkistonäkymän käytettävyyttä. Testaajan on teh-

tävä haku tietyn osaston kaikille työntekijöille annetulla päivämäärärajauksella, haettava tulosjoukosta tietty työntekijä ja tältä tietyt työpäivät. Tehtävässä käydään läpi myös kirjauksen poistaminen.

6.6.1 Ensimmäinen testaussessio

Ensimmäinen testaajista käytti vain hieman aikaa käyttöliittymään tutustumiseen ennen tehtävien aloittamista. Tästä huolimatta hän löysi heti kentän, jolla työntekijöitä voi suodattaa nimen perusteella, sekä ”Uusi kirjaus” -painikkeen. Testaaja syötti sisäänleimauksen kellonajan aluksi väärin, mutta huomasi virheen heti ja kommentoi tämän johtuvan testilaitteena käytetyn kannettavan tietokoneen näppäimistöstä.

Testaajalla oli hankaluuksia löytää keino suodattaa tietyn osaston työntekijät. Lisäksi testaajaa hämäsi se, että osastolista on esitetty hierarkkisena, vaikka ylimmän osaston valitseminen ei valitse sen alaosastoja eivätkä osaston työntekijät ole sen ylemmän osaston työntekijöiden alajoukko. Testaaja kommentoi myös, että olisi hyvä, jos osastohierarkian saisi piilotettua klikkaamalla elementin ulkopuolelle. Listan järjestäminen kokonaissaldon perusteella onnistui helposti.

Neljännessä tehtävässä testaaja löysi helposti oikean työntekijärivin, mutta ei ”Muokkaa saldoa” -painiketta. Testaaja kokeili klikata työntekijän päiväsaldon arvoa ja vei osoittimen seuraavaksi kokonaissaldoarvon kohdalle. Tästä testaajan huomio siirtyi edelleen viereiseen painikkeeseen, josta oikea toiminto löytyi.

Arkistonäkymän hakulomakkeella testaaja löysi työntekijöiden osastosuodatuksen nopeasti. Testaaja ei kuitenkaan huomannut, että työntekijärivit eivät ole oletuksena valittuina, ja hämmentyi, kun hakupainiketta ei pystynyt klikkaamaan. Testauksessa havaittiin myös virhetilanne, jossa Enter-painikkeen painaminen päivämäärän syötön jälkeen käynnisti haun, vaikka kaikkia pakollisia hakuehtoja ei oltu syötetty. Testaaja yritti ilmeisesti sulkea päivämääräkentän kalenterivalinnan Enter-painikkeella.

Testaaja löysi poikkeavassa osoitteessa tehdyn työaikakirjauksen helposti, mutta puutteellinen työpäivä löytyi hakutuloslistasta vasta toisella silmäilyllä.

6.6.2 Toinen testaussessio

Toinen testaaja tutustui käyttöliittymään huomattavasti ensimmäistä kauemmin. Testaaja arveli osoitesarakkaiden liittyvän työntekijän asuinpaikkaan sisään- ja ulosleimausten sijaan. Lisäksi hän piti epäselvänä ”Yhteenveto”-otsikkoa sekä ”Suodata nimellä” -tekstiä, ja ehdotti, että ”Hae nimellä” olisi kuvaavampi teksti.

Uutta kirjausta lisätessä testaaja syötti sisääntulon aikaleiman virheellisesti ilman tunteja ja minuutteja erottavaa kaksoispistettä. Testaaja huomasi virheen vasta yrittäessään tallentaa kirjausta, ja kommentoi, että virheellisten kenttien haaleanpunainen taustaväri saisi olla erottuvampi ja punaiset reunat kirkkaammat.

Testaaja löysi yhteenvetonäkymän osastosuodattimet heti, vaikkakin kommentoi, että pelkkä ”Listan suodattimet” olisi parempi nimi painikkeelle kuin ”Näytä listan suodattimet”, sillä näkymä on myös suodattimien muokkaamiseen, ei vain katseluun. Testaaja myös huomautti, että ”Tyhjennä kaikki” olisi loogisempi listan toiminto kuin ”Käännä valinta”, koska tällöin se sopisi paremmin yhteen viereisen ”Valitse kaikki”-toiminnon kanssa. Lisäksi yhden osaston valitseminen olisi sujuvampaa niin, että käyttäjä tyhjentää ensin kaikki valinnat ja valitsee sitten halutun osaston. Nykyisessä ratkaisussa optimaalinen toimintatapa on poistaa halutun osaston kohdalta valinta ja klikata sitten ”Käännä valinta” -painiketta. On epäintuitiivista, että osaston valitseminen tapahtuu perumalla valinta. Lisäksi testaaja ehdotti, että osastolistan sisennykset saisivat olla leveämmät, sillä nykyisellään näyttää siltä, että listan epätasainen asettelu johtuu virheellisestä tyylimäärittelystä eikä hierarkkisuudesta.

Kolmannessa tehtävässä testaaja hämmentyi siitä, että osastosuodatuksen asettamisen jälkeen yhteenvetonäkymän lista jäi tyhjäksi. Tämä johtui siitä, että nimisuodatus oli jäänyt päälle aiemmasta tehtävästä. Kun nimen suodatuskenttä on piilossa osastovalinnan elementin takana, ei käyttäjällä ole mitään tapaa nähdä, että listassa on nimisuodatus päällä, eikä nimellä suodattaminen ole välttämättä enää käyttäjän työmuistissa. Testaaja kommentoi myös, että nimisuodattimen tyhjentämiseen ei ole helppoa ja selkeää tapaa, sillä kentän teksti pitää pyyhkiä käsin.

Testaajalle oli selvää, että listan järjestäminen tapahtuu sarakeotsikoiden vieressä olevasta symbolista, mutta hän päätteli virheellisesti, että järjestyspainikkeita on yhden sijaan kaksi, eli ylös osoittava nuoli järjestäisi rivit suurimmasta saldomäärästä pienempään ja alanuoli päinvastoin.

Neljännessä tehtävässä testaaja oletti, että yhteenvetonäkymässä asetettu osastosuodatus pätisi myös arkistonäkymässä. Hän ei myöskään heti löytänyt vastaavaa

suodatusmahdollisuutta arkiston työntekijävalinnasta. Testaaja klikkasi vahingossa yhden työntekijän valituksi ja hämmentyi, kun haku palautti vain yhden työntekijärivin koko osaston sijaan. Lopuissa tehtävissä testaaja suoriutui hyvin, mutta huomautti, että myös puutteellisen työpäivärivin varoitusväri saisi olla erottuvampi.

6.6.3 Kolmas testaussessio

Kolmas testaaja kertoi käyttäneensä aiemmin työajanseurantasovellusta, joka oli tosin hyvin erinäköinen. Toisen testaajan tavoin hänkin käytti useita minuutteja käyttöliittymään tutustumiseen ja arveli osoitteiden olevan työntekijöiden kotiosoitteita. Hän myös toivoi suurempaa fonttikokoa.

Aikaisempien testaajien tapaan kolmaskin testaaja löysi nopeasti keinon hakea tietty työntekijä nimellä suodattamalla. Osastosuodatus ei kuitenkaan löytynyt heti, ja testaaja yrittikin ensin kirjoittaa osaston nimeä samaan kenttään, jolla työntekijän nimellä suodatus tapahtuu.

Osaston valinnan jälkeen testaaja ei ollut varma, päivittyikö työntekijälista automaattisesti vai pitäisikö hänen tehdä vielä jotain. Testaaja kysyi, näkeekö hän jostain, että nyt on vain tietyn osaston työntekijät näkyvissä, ja toivoi jonkinlaista indikaattoria voimassa olevista suodattimista.

Kirjausta syöttäessä testaaja ehdotti, että aikavalinta voisi tarjota alasvetovalikon, jossa olisi vaihtoehtoina kellonaikoja esimerkiksi puolen tunnin välein. Lisäksi hän esitti toiveen, että yhteenvetonäkymässä voisi säätää sarakkeiden leveyttä, jotta esimerkiksi vähemmän tärkeät osoitetiedot eivät veisi niin paljon tilaa.

Kuten ensimmäisessä testaussessiossa, ”Muokkaa saldoa” -painikkeen löytäminen tuotti ongelmia. Lisäksi saldon muokkauslomakkeella havaittiin virhe, jossa saldon tallennuspainike säilyi aktiivisena senkin jälkeen, kun saldomuutos oli tallennettu. Testaaja ei ollut varma, onnistuiko tallennus vai ei.

Toisen testaajan tavoin kolmas testaaja oletti, että arkistonäkymän haku tottelisi yhteenvetonäkymässä tehtyä osastovalintaa. Hän ei myöskään alkuun ymmärtänyt hakulomakkeen työntekijävalintaa ja sitä, miten tietyt kirjaukset saa haettua tietyiltä työntekijöiltä.

Kolmas testaaja poikkesi kahdesta aikaisemmasta siinä, että hän valitsi osastoja ja työntekijöitä klikkaamalla tekstiä valintaruudun sijaan. Aikaisemmat testaajat veivät aina osoittimen valintaruudun kohdalle huolimatta siitä, että koko rivi on klikattava.

6.6.4 Testauksessa havaitut puutteet

Taulukkoon 9 on koottu kolmessa testaussessiossa havaitut prototyypin käytettävyysongelmat. Kullekin puutteelle on määritetty painotus 1–3 sen mukaan, kuinka monen testaajan kohdalla se ilmeni, jolloin 3 tarkoittaa suurimman prioriteetin ongelmaa. Oikeimmanpuoleisessa sarakkeessa joko annetaan ehdotus ongelman korjaamiseksi käyttöliittymän jatkokehityksessä tai perustellaan, miksi ongelmaan ei tarvitse puuttua.

ongelma	**painotus**	**toimenpiteet tai perustelut**
Osastosuodatus on hankala löytää.	2	Muutetaan painikkeen teksti muotoon ”Listan suodattimet” ja tehdään siitä erottuvampi esimerkiksi suurentamalla fonttikokoa. Oletetaan myös, että painike on helpompi löytää, kun sen sijainnin on kerran oppinut.
Lomakkeita ei saa suljettua klikkaamalla niiden ulkopuolelle.	1	Toteuttamisesta saattaa aiheutua se, että käyttäjä sulkee lomakkeen vahingossa. Harkitaan toteutusta.
Tekstin koko on liian pieni.	1	Suurennetaan fonttia, mikäli varsinaisilta käyttäjiltä tulee vastaavaa palautetta.
”Muokkaa saldoa” -painike on hankala löytää.	2	Painike sijaitsee samassa paikassa kuin nykyisessä käyttöliittymässä, joten sen sijainti on tuttu nykyisille käyttäjille.
Arkistosta haettavien työntekijöiden valitseminen on epäselvää.	3	Esitetään selkeämmin tieto siitä, montako työntekijää on valittuna. Harkitaan, pitäisikö kaikkien työntekijöiden olla oletuksena valittuna.
Virheellisten kenttien ja puutteellisten työpäivien varoitusväri ei ole riittävän selkeä.	2	Tehdään varoitusväreistä tummempia ja kirkkaampia.
Ei ole selkeää, että ”Osoite”-tieto on nimenomaan sisään- tai ulosleimauksen osoite.	2	”Osoite”-otsikko on nykyisen käyttöliittymän mukainen. Selvitetään, ymmärtävätkö varsinaiset käyttäjät otsikon testaajia paremmin.
Sarakkeiden leveyttä ei voi säätää, jotta tärkeämmille tiedoille saisi enemmän tilaa.	1	Selvitetään mahdollisuudet toteuttaa sarakkeiden piilotus tai niiden leveyksien säätäminen.

”Tyhjennä kaikki” olisi loogisempi listatoiminto kuin ”Käännä valinta”.	1	Korvataan ”Käännä valinta”-toiminto kaikki valinnat tyhjentävällä toiminnolla.
Listasta ei suoraan näy, että osa riveistä on suodatettu.	2	Lisätään tieto rivien näkyvissä olevien rivien lukumäärästä sekä rivien kokonaismäärästä.
Nimen suodatuskentän tyhjentämiseen ei ole nopeaa tapaa.	1	Lisätään painike, jolla nimisuodatuksen saa tyhjennettyä.
Listan järjestyksen kääntämistä kuvaavan ikonin voi hahmottaa kahtena painikkeena yhden sijaan.	1	Näytetään painikkeella eri ulkoasu sen mukaan, onko lista järjestetty kyseisen sarakkeen mukaan laskevaan tai nousevaan järjestykseen.
Yhteenvetonäkymän osastovalinta ei päde arkistonäkymässä, vaikka käyttäjä olettaa niin.	2	Muutetaan osastovalintaelementti molemmille näkymille yhteiseksi tai synkronoidaan sen tila eri näkymien välillä.

Taulukko 9: Käytettävyystestauksessa havaitut puutteet

Arkiston hakulomakkeen työntekijävalinnan epäselvyys nousi esiin jokaisessa testaussessiossa, joten sitä on pyrittävä selkeyttämään jatkokehityksessä. Kahdella testaajista ilmeni ongelmia osastosuodatuksen ja saldon muokkauspainikkeen löytämisessä sekä ”Osoite”-sarakkeen ymmärtämisessä, mutta nämä ominaisuudet ovat todennäköisesti selkeämpiä kokeneemmille ja käyttöönoton ohjeistuksen saaneille käyttäjille, joten puutteet eivät välttämättä vaadi toimenpiteitä. Listan rivimäärän näyttäminen ja varoitusvärien korostaminen ovat helposti toteutettavia kehitysehdotuksia, jotka todennäköisesti parantavat käytettävyyttä. Myös kahden osastosuodattimen yhdistäminen on oletettavasti nykyistä parempi ratkaisu. Vain yhdessä testaussessiossa esiintyneisiin ongelmiin reagoidaan, mikäli työmäärä ei ole suuri ja toteutuksella ei ole nähtävissä haittapuolia. Esimerkiksi fonttikoon kasvattaminen tehdään harkitusti, sillä se voi vaikuttaa ulkoasun rakenteeseen.

Käytettävyysongelmien lisäksi testauksessa havaittiin kaksi ohjelmointivirhettä – saldolomakkeen tallennuspainikkeen jääminen aktiiviseksi tallennuksen jälkeen ja puutteellisen arkistohaun käynnistäminen Enter-painikkeella – jotka korjataan joka tapauksessa.

Jokaisen testaajan kohdalla ilmeni hämmennystä siitä, että uusi työaikakirjaus ei

ilmestynyt näkyviin yhteenvetonäkymään tai hakutuloslistaan tallennuksen jälkeen. Puute johtuu siitä, ettei prototyypin mock-rajapintatoteutusta koettu kannattavaksi kehittää niin pitkälle, että uusi kirjausrivi lisättäisiin oikeasti tietomassaan. Tämä olisi kuitenkin tehnyt käytettävyystestauksesta sujuvampaa ja simuloinut todellisen sovelluksen käyttöä paremmin.

7 Yhteenveto ja johtopäätökset

Ammattikäyttöön tarkoitetun toimistosovelluksen tapauksessa käytettävyys tarkoittaa pääasiassa sitä, miten tehokkaasti käyttäjät pystyvät tekemään työtänsä sovelluksen puitteissa. Sovelluksen käyttöliittymän pitää olla helposti opittava, jotta käyttäjä pystyy aloittamaan työnteon nopeasti. Virheiden tekemisen mahdollisuus on minimoitava, ja mikäli virhe tapahtuu, on käyttäjän pystyttävä huomaamaan ja korjaamaan se. Sovelluksen käytöstä saatu mielihyvä on toimistosovellusten tapauksessa toissijaista verrattuna vapaa-ajan sovelluksiin, mutta käyttäjien tyytyväisyys saattaa kuitenkin vaikuttaa siihen, pysyykö sovellus jatkossakin yrityksen käytössä.

Havaitsemista, oppimista ja ajattelua tutkivasta kognitiotieteestä on hyötyä ymmärrettävän käyttöliittymän suunnittelussa. Elementtien sijoittelu, muotoilu, värimaailma, symboliikka ja käyttäytyminen määräävät, miten käyttäjä hahmottaa käyttöliittymän. Aikaisemmat kokemukset ja mielikuvat muista käyttöliittymistä sekä ympäröivästä maailmasta yleisesti vaikuttavat nekin käyttäjän tapaan hahmottaa näkemäänsä. Vaikeudet käyttöliittymän ymmärtämisessä johtavat hitaampaan ja tehottomampaan työntekoon ja pahimmassa tapauksessa virheisiin tai jopa työnteon estymiseen.

Tämän tutkielman tavoitteena oli kartoittaa toimeksiantajayrityksen työaikasovelluksen nykyisen käyttöliittymän ongelmakohdat ja tarjota niihin ratkaisu suunnittelemalla ja kehittämällä korvaavan käyttöliittymän prototyyppi. Suurimpana nykyisen sovelluksen ongelmana on sen listanäkymien skaalautumattomuus suurelle tietomäärälle, mutta myös pienempiä puutteita havaittiin runsaasti. Prototyyppi toteutettiin vaatimusmäärittelyn ja käyttöliittymäluonnosten pohjalta yksisivuisena sovelluksena, jonka pääteknologiaksi valittiin Vue.js sen uutuuden, kasvavan suosion ja kehutun opittavuuden takia. Automaattisella testauksella ja koodin analysoinnilla pyritään huolehtimaan lähdekoodin ylläpidettävyydestä, jotta uusien virhekorjausten, käytettävyysparannusten ja uusien ominaisuuksien kehittäminen on jatkossa helppoa.

Valmis prototyyppi toteuttaa lähes kaikki määritellyt käyttötapaukset, ja puuttumaan jäi ainoastaan yksi pienemmän prioriteetin ominaisuus, jonka työmäärää ei osattu arvioida suunnitteluvaiheessa riittävän hyvin. Prototyyppi todettiin käyttökelpoiseksi työpöytälaitteella neljällä eri selaimella sekä tabletilla, joskin mobiililaitteilla käytettävyys on heikompi.

Vasteaikatestauksessa ilmeni, että joissain toiminnoissa odotusaika saattaa olla koh-

tuuttoman pitkä, mikäli työntekijärivejä on useita tuhansia. Prototyyppi kuitenkin suoriutui monista toiminnoista kelvollisessa ajassa suuresta tietomäärästä huolimatta. Sovellusten nykyisten käyttäjien tietomäärillä vasteajat pysyvät todennäköisesti hyvinä, mutta lisäoptimointi saattaa tulla kyseeseen käyttäjäkunnan laajentuessa.

Koodin ylläpidettävyydestä on pyritty huolehtimaan automaatiotestauksella sekä staattisella koodin analysoinnilla, joista kumpaakaan ei nykyisessä käyttöliittymätoteutuksessa ole. Myös JavaScript-sovelluskehys on valittu osittain sillä perusteella, että sitä on useassa lähteessä kuvailtu helposti opittavaksi ja pienille yrityksille sopivaksi. Tämän lisäksi on kuitenkin jatkossa hyödynnettävä koodin vertaisarviointia, jotta yrityksen kehittäjistä muutkin kuin tekijä ymmärtävät käyttöliittymän lähdekoodia ja pystyvät ylläpitämään sekä kehittämään sitä.

Käytettävyystestauksessa nousi esille useita parannusehdotuksia, jotka otetaan huomioon käyttöliittymän jatkokehityksessä. Osa testaajien kohtaamista ongelmista on perusteltavissa sillä, että testiryhmä ei edustanut sovelluksen todellista käyttäjäryhmää eikä heillä ollut juurikaan käyttöohjeistusta tai kokemusta työajanseurantasovelluksen nykyisestä käyttöliittymästä, jota prototyypin ratkaisut osittain mukailevat. Toisaalta nykyisten käyttöliittymäratkaisujen soveltaminen sellaisenaan ei ole aina hyvä valinta, sillä tuttu ja turvallinen vaihtoehto ei välttämättä tarkoita parasta mahdollista vaihtoehtoa.

Prototyyppi tarjoaa nykyistä modernimman ja helpommin ylläpidettävän käyttöliittymätoteutuksen. Uusi käyttöliittymä tarjoaa samat toiminnot kuin nykyinenkin ja pystyy tarvittaessa esittämään jopa 5 000 työntekijän tiedot niin, että tietyn työntekijän löytäminen tietomassasta on edelleen mahdollista. Lisäksi uusi käyttöliittymä tarjoaa ratkaisut useisiin nykyisen käyttöliittymän pienempiin käytettävyysongelmiin, kuten elementtien hämäävään tyylittelyyn. Voidaan siis todeta, että prototyyppi on onnistunut tavoitteissaan ja että sillä voidaan tulevaisuudessa korvata nykyinen käyttöliittymä.

Ajallisten resurssien puutteessa varsinaisten loppukäyttäjien näkemys prototyypistä jäi selvittämättä tämän tutkielman puitteissa. Näin ollen tutkielman tekijä ehdottaa, että työaikasovelluksen uudistusprojektin seuraavana askeleena toteutetaan ne käytettävyystestauksessa havaitut puutteet, jotka todetaan tässä vaiheessa kannattaviksi, ja tämän jälkeen prototyyppi esitellään loppukäyttäjän edustajille.

Lähteet

Agg18 Aggarwal, S., Modern Web-Development using ReactJS. *International Journal of Recent Research Aspects*, 5, sivut 133–137.

All18 Allen, I., The Brutal Lifecycle of Javascript Frameworks, 2018. `https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/`. [24.3.2018]

ang18a Angular Docs, 2018. `https://angular.io/docs`. [26.5.2018]

ang18b AngularJS Developer Guide, 2018. `https://docs.angularjs.org/guide`. [10.3.2018]

Bab16 Babich, N., UX: Infinite Scrolling vs. Pagination, 2016. `https://uxplanet.org/ux-infinite-scrolling-vs-pagination-1030d29376f1`. [23.9.2018]

bab18 What is Babel?, 2018. `https://babeljs.io/docs/en/`. [26.3.2019]

BB13 Baturay, M. ja Birtane, M., Responsive Web Design: A New Type of Design for Web-based Instructional Content. *Procedia - Social and Behavioral Sciences*, 106, sivut 2275–2279.

BGR17 Benitte, R., Greif, S. ja Rambeau, M., The State of Developer Ecosystem in 2017, 2017. `https://www.jetbrains.com/research/devecosystem-2017/`. [11.4.2018]

bro18 Browser Market Share, 2018. `https://netmarketshare.com/browser-market-share.aspx`. [14.7.2018]

Cre13 Crespo, G., *Responsive web design with jQuery*. Packt Publishing, Birmingham, Iso-Britannia, 2013.

Dac17 Dace, Understanding React's Virtual DOM vs. the real DOM, 2017. `https://medium.com/@hidace/understanding-reacts-virtual-dom-vs-the-real-dom-68ae29039951`. [29.3.2018]

Dou18 Douglas, S., 5 best practices for getting modal windows right, 2018. `https://www.justinmind.com/blog/5-best-practices-for-getting-modal-windows-right/`. [10.11.2018]

ecm15 ECMAScript 2015 Language Specification, 2015. https://www.ecma-international.org/ecma-262/6.0/. [26.3.2019]

Eva17 Evans, D., *Bottlenecks: Aligning UX Design with User Psychology*. Apress, Yhdysvallat, 2017.

Fes17 Fessenden, T., Modal & Nonmodal Dialogs: When (& When Not) to Use Them, 2017. https://www.nngroup.com/articles/modal-nonmodal-dialog/. [10.11.2018]

fus18 Top JavaScript Frontend Frameworks Comparison in 2018, 2018. https://www.fusioncharts.com/resources/js-frontend-frameworks-comparison/. [11.4.2018]

Gar05 Garrett, J., Ajax: A New Approach to Web Applications, 2005. https://www.adaptivepath.org/ideas/ajax-new-approach-web-applications/. [29.3.2019]

GB16 Germanakos, P. ja Belk, M., *Human-Centred Web Adaptation and Personalization: From Theory to Practice*. Springer, Sveitsi, 2016.

GGRR17 Grigera, J., Garrido, A., Rivero, J. ja Rossi, G., Automatic detection of usability smells in web applications. *International Journal of Human-Computer Studies*, 97, sivut 129–148.

Gim16 Gimeno, A., How JavaScript bundlers work, 2016. https://medium.com/@gimenete/how-javascript-bundlers-work-1fc0d0caf2da. [4.1.2019]

Gre16 Greene, E., Two-Way Data Binding: Angular 2 and React, 2016. https://www.accelebrate.com/blog/two-way-data-binding-angular-2-and-react/. [24.3.2018]

HMPR04 Hevner, A., March, S., Park, J. ja Ram, S., Design Science in Information Systems Research. *MIS Quarterly*, 28, sivut 75–105.

iso98 ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. Tekninen raportti, International Organization for Standardization, maaliskuu 1998.

Jac19 Jackson, C., The perils of using Internet Explorer as your default browser, 2019. https://techcommunity.microsoft.com/t5/Windows-IT-Pro-Blog/The-perils-of-using-Internet-Explorer-as-your-default-browser/ba-p/331732. [26.3.2019]

Joh14 Johnson, J., *Designing with the mind in mind : simple guide to understanding user interface design guidelines.* Morgan Kaufmann, Amsterdam, Alankomaat, 2014.

Jon02 Jones, N., Introduction to Lint. *Embedded Systems Programming*, sivu 55.

KB18 Kaimer, F. ja Brune, P., Return of the JS: Towards a Node.js-Based Software Architecture for Combined CMS/CRM Applications. *Procedia Computer Science*, 141, sivut 454–459.

KKGF16 Kiruthika, J., Khaddaj, S., Greenhill, D. ja Francik, J., User Experience design in web applications. *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, sivut 642–646.

Kof35 Koffka, K., *Principles of Gestalt Psychology.* Routledge, Abingdon, Iso-Britannia, 1935.

Kra15 Krajka, B., The difference between Virtual DOM and DOM, 2015. http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/. [29.3.2018]

Kra17 Krause, S., Results for js web frameworks benchmark - round 6, 2017. http://www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html. [25.3.2018]

Leh13 Lehdonvirta, P. ja Korpela, J., *HTML5 sovellusalustana.* RPS-yhtiöt, Helsinki, 2013.

LHH14 Lestari, D., Hardianto, D. ja Hidayanto, A., Analysis of User Experience Quality on Responsive Web Design from its Informative Perspective. *International Journal of Software Engineering and Its Applications*, 8, sivut 642–646.

Mal17 Malhotra, M., Vue.js Is Good, But Is It Better Than Angular Or React?, 2017. https://www.valuecoders.com/blog/technology-and-apps/vue-js-comparison-angular-react/#. [29.3.2018]

Mat14 Mathis, L., Design Patterns for Replacing Modal Windows, 2014. https://community.appway.com/screen/kb/article/design-patterns-for-replacing-modal-windows-1482810903553. [25.11.2018]

MN17 Mlynarski, A. ja Nurzynska, K., Comparative analysis of JavaScript and its extensions for web application optimization. *Communications in Computer and Information Science*, 716, sivut 539–550.

Moh13 Mohorovičić, S., Implementing responsive web design for enhanced web presence. *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, sivut 1206–1210.

Neu17 Neuhaus, J., Angular vs. React vs. Vue: A 2017 comparison, 2017. https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176. [24.3.2018]

Nie94 Nielsen, J., *Usability Engineering*. Morgan Kauffman Publishers Inc, 1994.

Nie06 Nielsen, J., F-Shaped Pattern For Reading Web Content, 2006. https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content-discovered/. [13.4.2019]

Nie08 Nielsen, J., OK-Cancel or Cancel-OK? The Trouble With Buttons, 2008. https://www.nngroup.com/articles/ok-cancel-or-cancel-ok/. [22.9.2018]

Nie12 Nielsen, J., How Many Test Users in a Usability Study?, 2012. https://www.nngroup.com/articles/how-many-test-users/. [14.4.2019]

Pet18 Petrosyan, M., Angular 5 vs. React vs. Vue, 2018. https://itnext.io/angular-5-vs-react-vs-vue-6b976a3f9172. [29.3.2018]

PGA18 Pano, A., Graziotin, D. ja Abrahamsson, P., Factors and actors leading to the adoption of a JavaScript framework, 2018.

Pie15 Pierce, P., When to paginate and when to infinite scroll, 2015. https://www.creativebloq.com/ux/paginate-or-infinite-scroll-71515816. [23.9.2018]

Pra17 Prajapati, D., Infinite Scrolling Vs. Pagination, 2017. https://medium.com/@dalpattapaniya/infinite-scrolling-vs-pagination-f237592f339a. [23.9.2018]

PZL08 Pautasso, C., Zimmermann, O. ja Leymann, F., RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. *17th International World Wide Web Conference*, sivut 805–814.

QF16 Qiang, S. ja Fei, H., An Icon Design Approach Based on Symbolic and Users' Cognitive Psychology. *Indonesian Journal of Electrical Engineering and Computer Science*, 4, sivut 695–705.

Rai17 Rai, H., Moving From Angular to Vue : A vuetiful journey, 2017. https://medium.com/@Hemantisme/moving-from-angular-to-vue-a-tiful-journey-c29842ab2039. [25.3.2018]

rea18 React Docs, 2018. https://reactjs.org/docs/. [24.3.2018]

Ruf14 Rufus, V., *AngularJS web application development blueprints : a pratical guide to developing powerful web applications with AngularJS*. Packt Publishing, Birmingham, Iso-Britannia, 2014.

Sha17 Shahzad, F., Modern and Responsive Mobile-enabled Web Applications. *Procedia Computer Science*, 110, sivut 410–415.

sta17 The State of JavaScript 2017, 2017. https://stateofjs.com/2017/front-end/results/. [11.4.2018]

sta18 The State of JavaScript 2018, 2018. https://2018.stateofjs.com/front-end-frameworks/overview/. [4.1.2019]

sta19 Stack Overflow Trends, 2019. https://insights.stackoverflow.com/trends. [13.4.2019]

sur17 Developer Survey Results 2017, 2017. https://insights.stackoverflow.com/survey/2017. [11.4.2018]

TA13 Tullis, T. ja Albert, W., *Measuring the User Experience : Collecting, Analyzing and Presenting Usability Metrics*. Elsevier, 2013.

TB14 Tarasiewics, P. ja Böhm, R., *AngularJS*. Brainy Software, Heidelberg, Saksa, 2014.

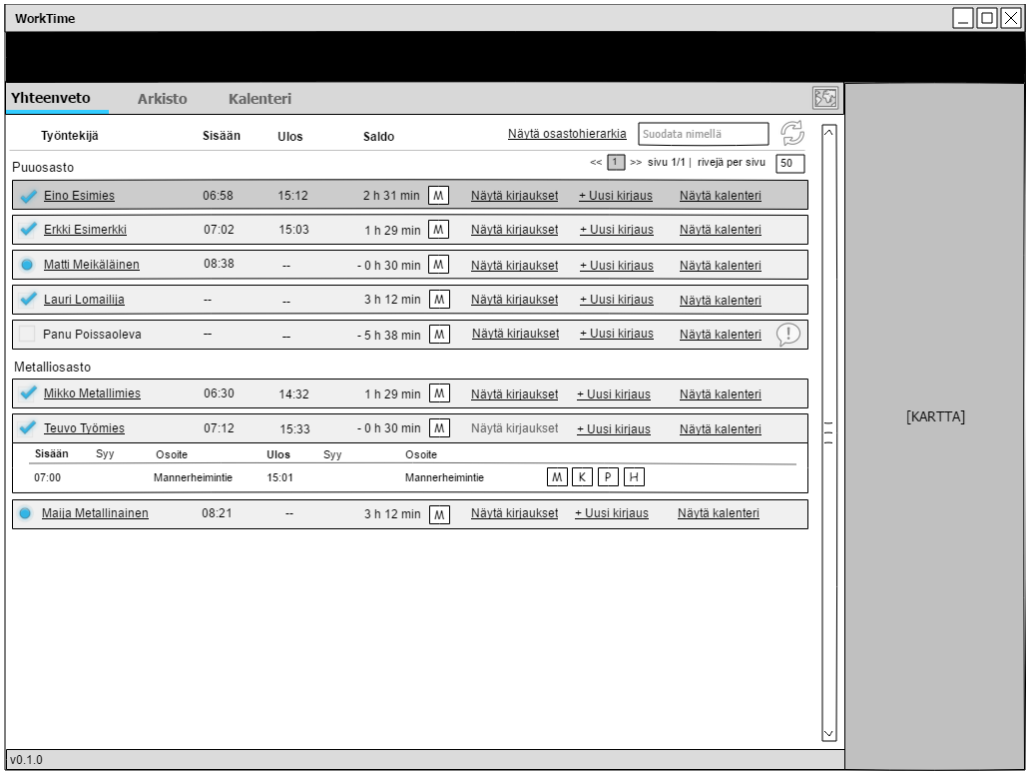
vue16 Retiring vue-resource, 2016. `https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4`. [10.1.2019]

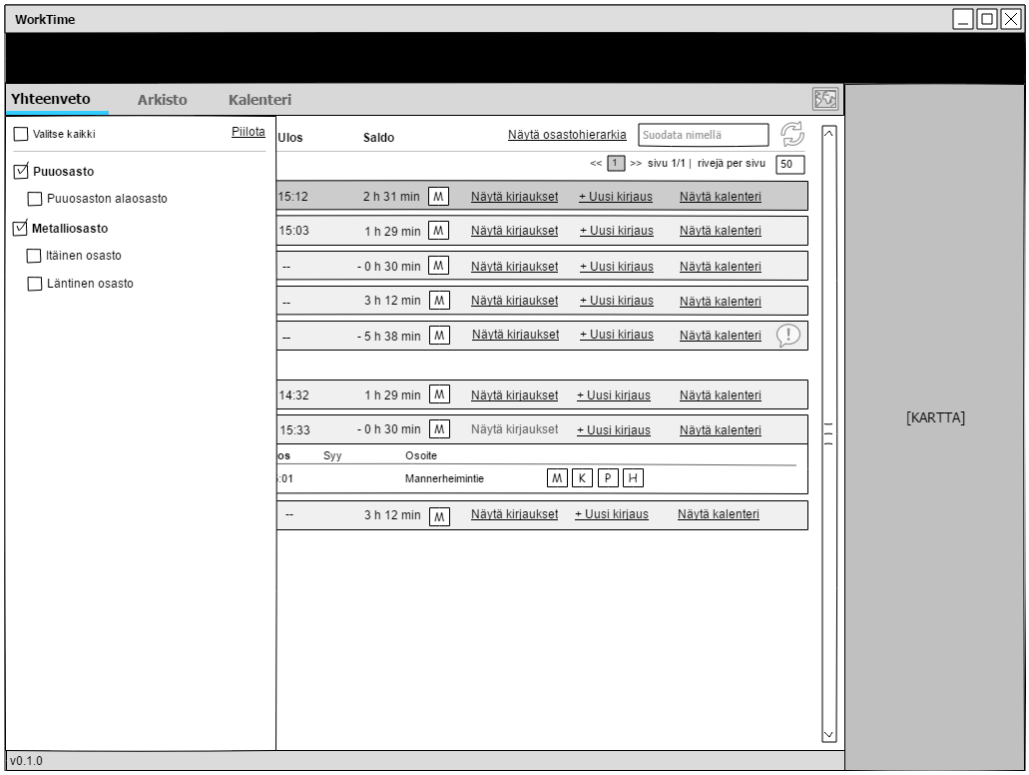
vue18 Vue Test Utils, 2018. `https://vue-test-utils.vuejs.org/`. [19.1.2019]

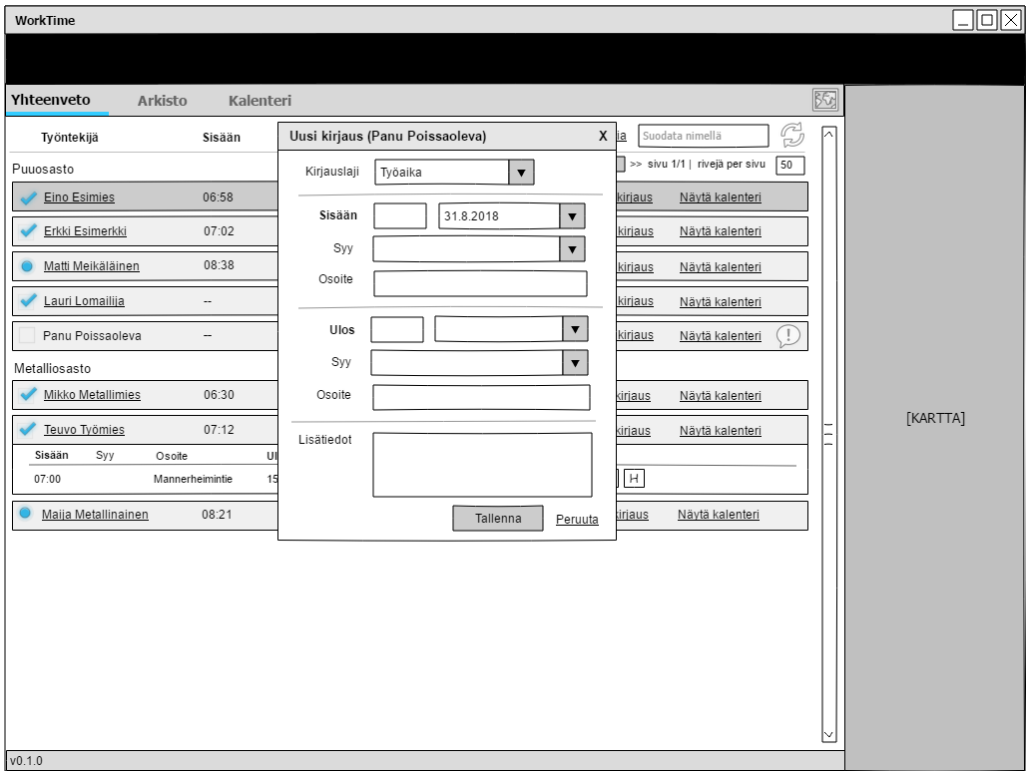
vue19 Introduction - Vue.js, 2019. `https://vuejs.org/v2/guide/`. [4.1.2019]

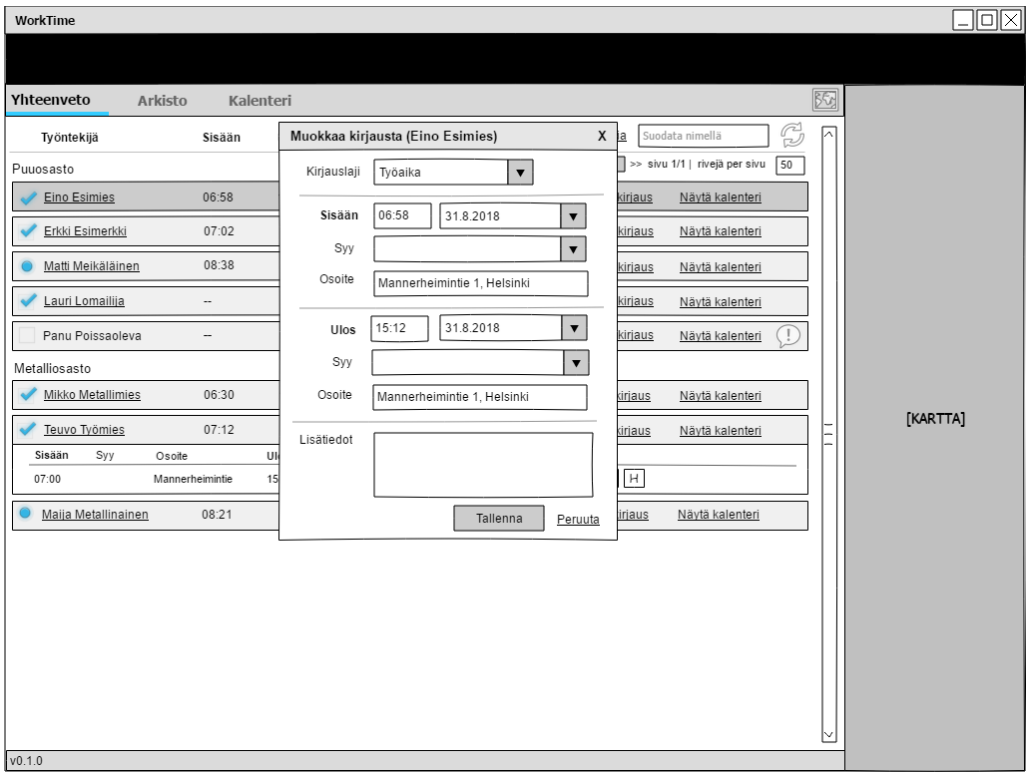
YLYZ12 Yee, C., Ling, C., Yee, W. ja Zainon, W., GUI Design Based on Cognitive Psychology: Theoretical, Empirical and Practical Approaches, 2012.

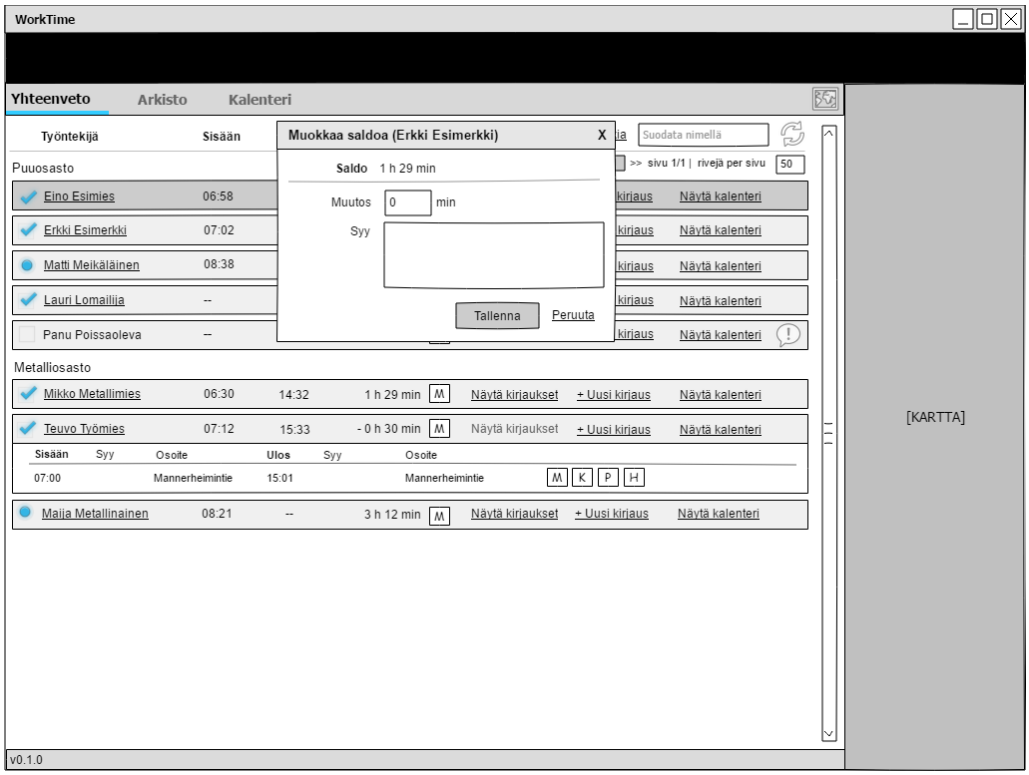
Liite 1. Käyttöliittymäluonnokset

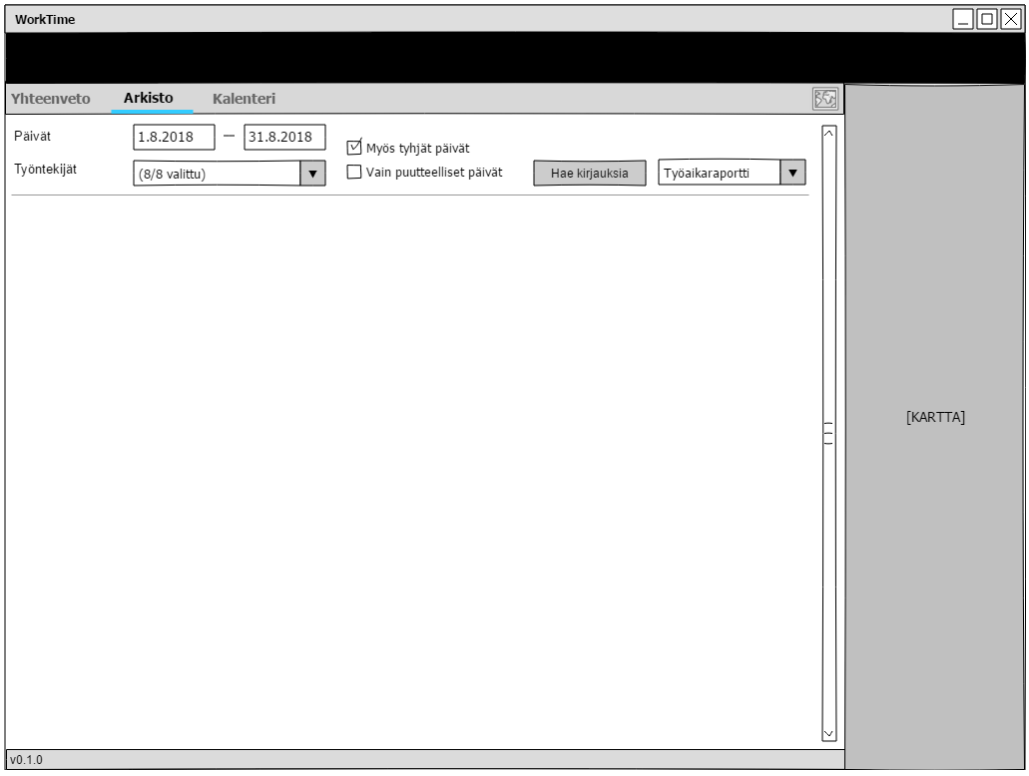


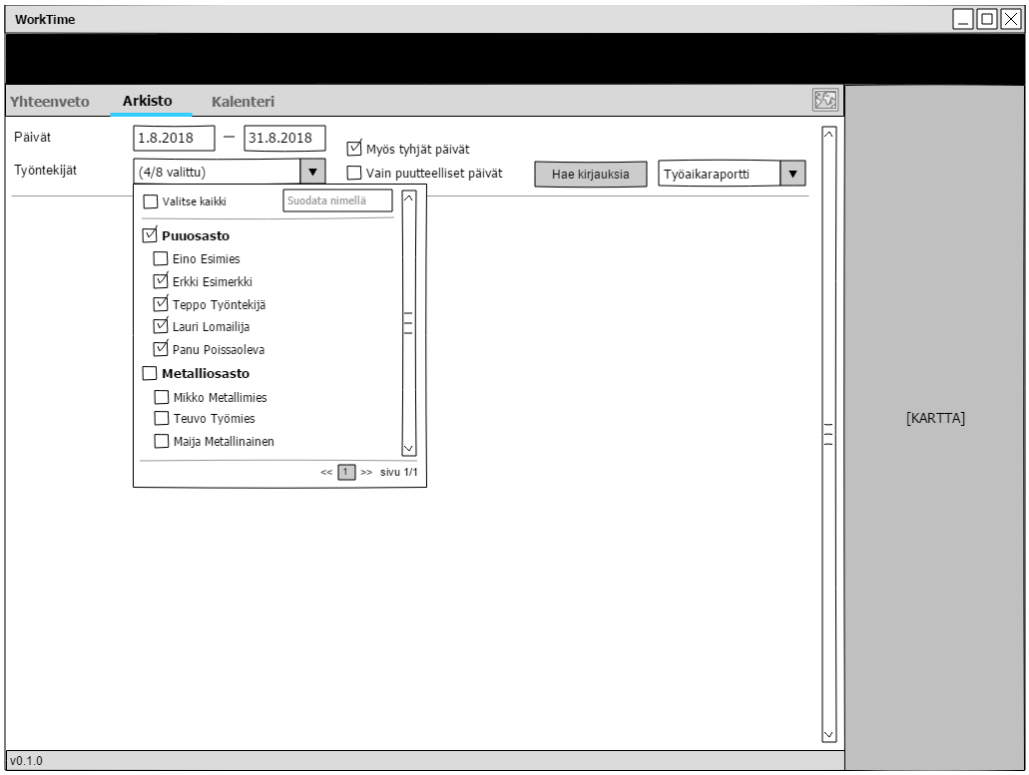


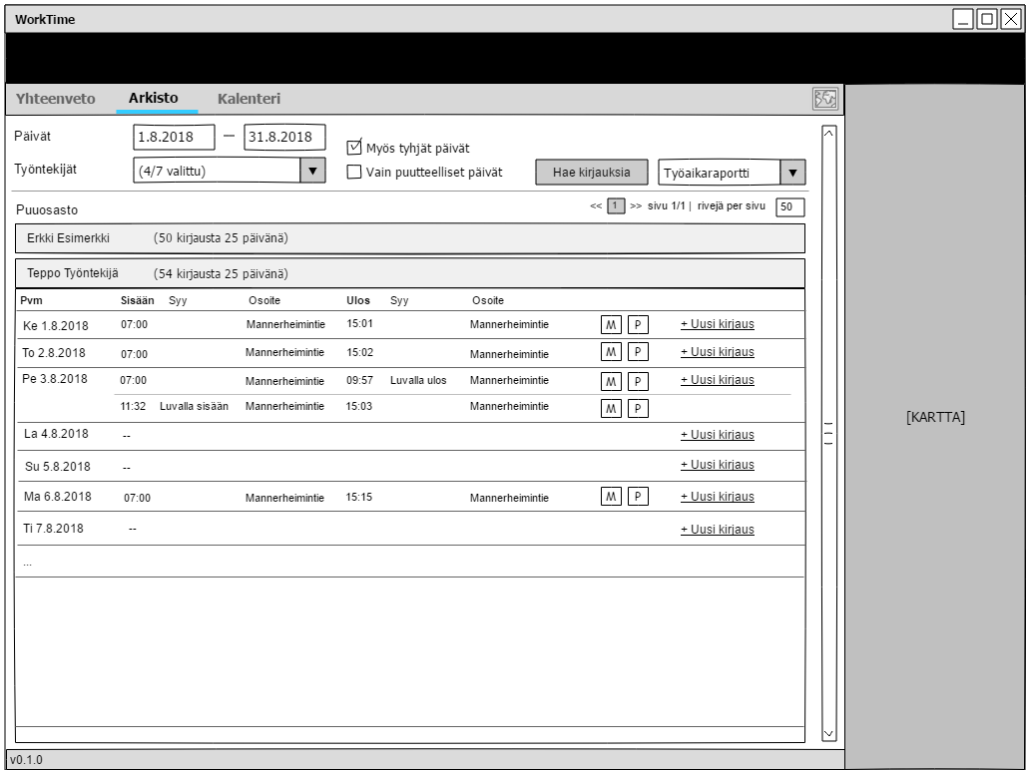

WorkTime
Yhteenveto
Arkisto
Kalenteri
Työntekijä
Sisään
Puuosasto
Eino Esimies 06:58
Erkki Esimerkki 07:02
Matti Meikäläinen 08:38
Lauri Lomailija --
Panu Poissaoleva --
Metalliosasto
Mikko Metallimies 06:30
Teuvo Työmies 07:12
Sisään Syy Osoite
07:00 Mannerheimintie
Maija Metallinainen 08:21
Uusi kirjaus (Panu Poissaoleva)
Kirjauslaji Työaika
Sisään 31.8.2018
Syy
Osoite
Ulos
Syy
Osoite
Lisätiedot
Tallenna
Peruuta
Suodata nimellä
>> sivu 1/1 | rivejä per sivu 50
kirjaus Näytä kalenteri
[KARTTA]
v0.1.0

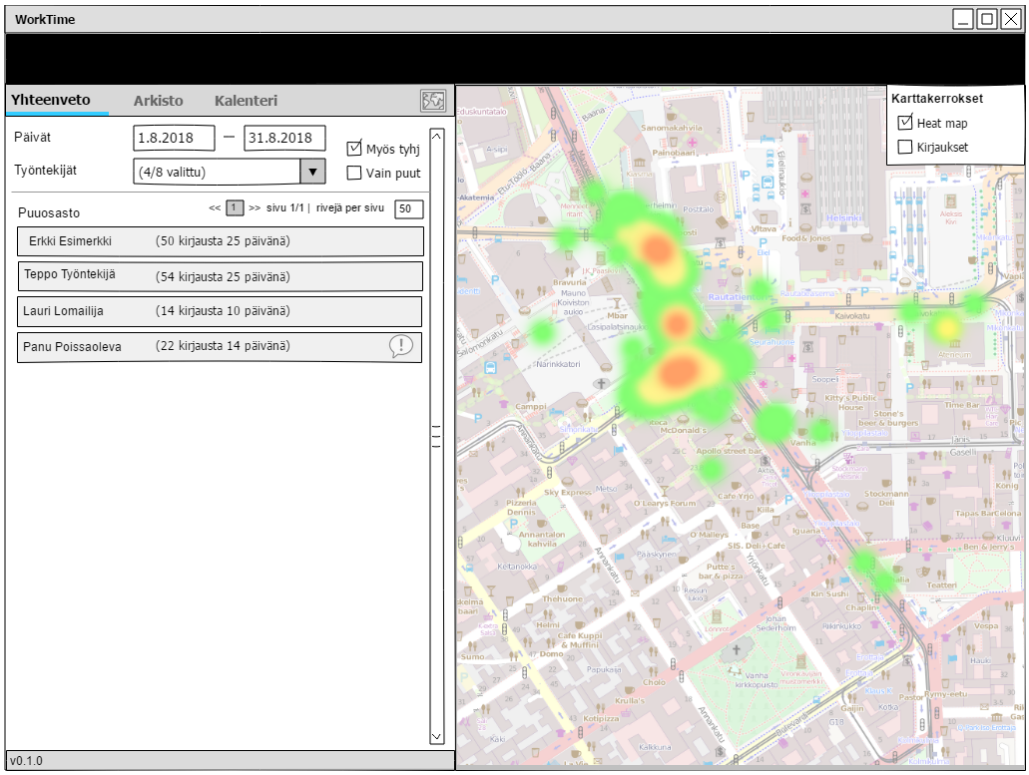

WorkTime
Yhteenveto
Arkisto
Kalenteri
Työntekijä
Sisään
Puuosasto
Eino Esimies 06:58
Erkki Esimerkki 07:02
Matti Meikäläinen 08:38
Lauri Lomailija --
Panu Poissaoleva --
Metalliosasto
Mikko Metallimies 06:30
Teuvo Työmies 07:12
Sisään Syy Osoite
07:00 Mannerheimintie
Maija Metallinainen 08:21
Muokkaa kirjausta (Eino Esimies)
Kirjauslaji Työaika
Sisään 06:58 31.8.2018
Syy
Osoite Mannerheimintie 1, Helsinki
Ulos 15:12 31.8.2018
Syy
Osoite Mannerheimintie 1, Helsinki
Lisätiedot
Tallenna
Peruuta
Suodata nimellä
>> sivu 1/1 | rivejä per sivu 50
kirjaus Näytä kalenteri
[KARTTA]
v0.1.0

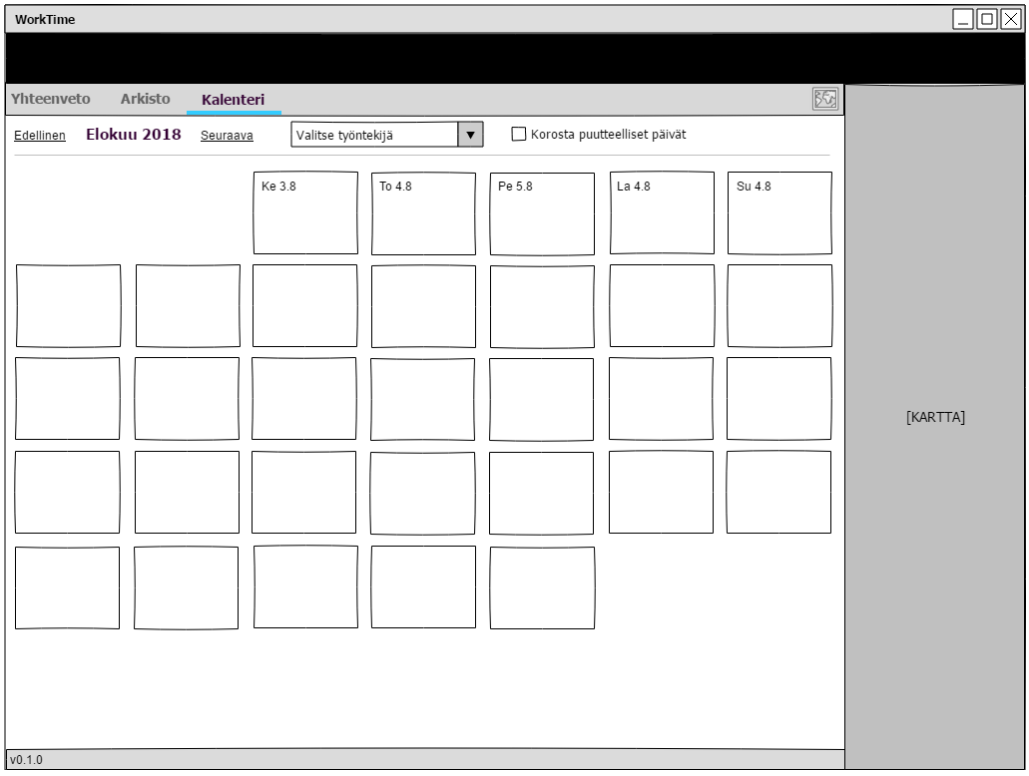

WorkTime
Yhteenveto
Arkisto
Kalenteri
Työntekijä
Sisään
Muokkaa saldoa (Erkki Esimerkki)
X
Saldo 1 h 29 min
Muutos 0 min
Syy
Tallenna
Peruuta
Suodata nimellä
>> sivu 1/1 | rivejä per sivu 50
Puuosasto
Eino Esimies 06:58
Erkki Esimerkki 07:02
Matti Meikäläinen 08:38
Lauri Lomailija --
Panu Poissaoleva --
Näytä kalenteri
Metalliosasto
Mikko Metallimies 06:30 14:32 1 h 29 min
Teuvo Työmies 07:12 15:33 - 0 h 30 min
Näytä kirjaukset
+ Uusi kirjaus
Sisään Syy Osoite Ulos Syy Osoite
07:00 Mannerheimintie 15:01 Mannerheimintie
M K P H
Maija Metallinainen 08:21 -- 3 h 12 min
[KARTTA]
v0.1.0


WorkTime
Yhteenveto
Arkisto
Kalenteri
Päivät
1.8.2018
31.8.2018
Työntekijät
(8/8 valittu)
Myös tyhjät päivät
Vain puutteelliset päivät
Hae kirjauksia
Työaikaraportti
[KARTTA]
v0.1.0


WorkTime
Yhteenveto
Arkisto
Kalenteri
Päivät
1.8.2018
31.8.2018
Työntekijät
(4/8 valittu)
Myös tyhjät päivät
Vain puutteelliset päivät
Hae kirjauksia
Työaikaraportti
Valitse kaikki
Suodata nimellä
Puuosasto
Eino Esimies
Erkki Esimerkki
Teppo Työntekijä
Lauri Lomailija
Panu Poissaoleva
Metalliosasto
Mikko Metallimies
Teuvo Työmies
Maija Metallinainen
<< 1 >> sivu 1/1
[KARTTA]
v0.1.0


WorkTime
Yhteenveto
Arkisto
Kalenteri
Päivät
1.8.2018
31.8.2018
Työntekijät
(4/7 valittu)
Myös tyhjät päivät
Vain puutteelliset päivät
Hae kirjauksia
Työaikaraportti
Puuosasto
<< 1 >> sivu 1/1 | rivejä per sivu 50
Erkki Esimerkki (50 kirjausta 25 päivänä)
Teppo Työntekijä (54 kirjausta 25 päivänä)
Pvm Sisään Syy Osoite Ulos Syy Osoite
Ke 1.8.2018 07:00 Mannerheimintie 15:01 Mannerheimintie M P + Uusi kirjaus
To 2.8.2018 07:00 Mannerheimintie 15:02 Mannerheimintie M P + Uusi kirjaus
Pe 3.8.2018 07:00 Mannerheimintie 09:57 Luvalla ulos Mannerheimintie M P + Uusi kirjaus
11:32 Luvalla sisään Mannerheimintie 15:03 Mannerheimintie M P
La 4.8.2018 -- + Uusi kirjaus
Su 5.8.2018 -- + Uusi kirjaus
Ma 6.8.2018 07:00 Mannerheimintie 15:15 Mannerheimintie M P + Uusi kirjaus
Ti 7.8.2018 -- + Uusi kirjaus
...
[KARTTA]
v0.1.0


WorkTime
Yhteenveto
Arkisto
Kalenteri
Päivät
1.8.2018
31.8.2018
Myös tyhj
Työntekijät
(4/8 valittu)
Vain puut
Puuosasto
<< 1 >> sivu 1/1 | rivejä per sivu 50
Erkki Esimerkki (50 kirjausta 25 päivänä)
Teppo Työntekijä (54 kirjausta 25 päivänä)
Lauri Lomailija (14 kirjausta 10 päivänä)
Panu Poissaoleva (22 kirjausta 14 päivänä)
v0.1.0
Karttakerrokset
Heat map
Kirjaukset


WorkTime
Yhteenveto
Arkisto
Kalenteri
Edellinen
Elokuu 2018
Seuraava
Valitse työntekijä
Korosta puutteelliset päivät
Ke 3.8
To 4.8
Pe 5.8
La 4.8
Su 4.8
v0.1.0
[KARTTA]

Liite 2. Automaatiotestien kattavuusraportti

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
src	100	100	100	100	
filters.js	100	100	100	100	
utils.js	100	100	100	100	
src/components	100	100	100	100	
app.vue	100	100	100	100	
header.vue	100	100	100	100	
main-content.vue	100	100	100	100	
map.vue	100	100	100	100	
resizer.vue	100	100	100	100	
src/components/archive	100	100	100	100	
archive.vue	100	100	100	100	
day-row.vue	100	100	100	100	
search-form.vue	100	100	100	100	
worker-list-header.vue	100	100	100	100	
worker-row.vue	100	100	100	100	
src/components/calendar	100	100	100	100	
calendar-view.vue	100	100	100	100	
calendar.vue	100	100	100	100	
src/components/forms	100	100	100	100	
balance-form.vue	100	100	100	100	
confirm-modal.vue	100	100	100	100	
datetime.vue	100	100	100	100	
form-item.vue	100	100	100	100	
record-form.vue	100	100	100	100	
src/components/list	100	100	100	100	
pagination.vue	100	100	100	100	
record-row.vue	100	100	100	100	
worker-list-header.vue	100	100	100	100	
worker-list.vue	100	100	100	100	
worker-row.vue	100	100	100	100	
src/components/resource-lists	100	100	100	100	
hierarchy-row.vue	100	100	100	100	
hierarchy.vue	100	100	100	100	
worker-list-header.vue	100	100	100	100	
worker-row.vue	100	100	100	100	
src/components/summary	100	100	100	100	
summary-settings.vue	100	100	100	100	
summary.vue	100	100	100	100	
worker-list-header.vue	100	100	100	100	
worker-records.vue	100	100	100	100	
worker-row.vue	100	100	100	100	

```
Test Suites: 30 passed, 30 total
Tests:       138 passed, 138 total
Snapshots:   0 total
Time:        18.025s
Ran all test suites.
```

Liite 3. Käytettävyystestauksen ohje

Työaikasovelluksen käytettävyystestaus

Tämä hallintakäyttöliittymä mahdollistaa työntekijöiden työaikakirjausten – eli sisään- ja ulosleimausten – katselun ja muokkaamisen. Olet esimies, jolla on päivitysoikeus yhteensä 5 000 työntekijän tietoihin. Sinulle on tarjolla kolme eri näkymää: kuluvan päivän yhteenveto, arkisto sekä beeta-vaiheessa oleva kalenteri.

1. Tutustu käyttöliittymään päällisin puolin.

2. Lisää työntekijälle Sarah Wickins uusi työaikakirjaus tälle päivälle. Sarah kirjautui sisään klo 16.00 ja ulos klo 17.15. Laita kirjauksen lisätiedoksi ”Unohti leimata sisään”.

3. Kenelle Puuosaston työntekijälle on kertynyt eniten kokonaissaldoa?

4. Kuukauden työntekijä on ansainnut ylimääräistä saldoa 45 minuuttia. Tiedät hänen kuuluvan Metsäosastoon ja muistat sukunimen olevan jokin worth-päätteinen. Tiedät myös sen, että hän ei ole leimannut itseään ulos vielä tänään. Lisää tälle työntekijälle 45 minuuttia saldoa lisätiedolla ”Kuukauden työntekijä”.

5. Hae kaikkien Ympäristöosaston työntekijöiden kirjaukset ajalta 28.1.2019–3.3.2019. Osastoon kuuluu reilu 700 työntekijää.
   a) Valitse haetuista työntekijöistä se, jonka kokonaissaldo on eniten miinuksella.
   b) Etsi yllä mainitulta työntekijältä päivä, jolta puuttuu työaikakirjaus. Lisää tälle päivälle uusi kirjaus.
   c) Yllä mainittu työntekijä on helmikuun viimeisellä viikolla leimannut itsensä sisään Mannerheimintien työmaan ulkopuolella. Poista kyseinen kirjaus.